LEARN MYSQL

什麼是 MYSQL

MySQL？名字有什麼意義？

MySQL 的名字來自其共同創辦人之一 Monty Widenius 的女兒 My。 "My"和"SQL"組合起來就是 MySQL。

MySQL 是一款強大的資料庫管理系統，專為管理關聯式資料庫而設計。它是 **Oracle 支援的開源軟體**，這意味著您可以免費使用 MySQL。此外，您還可以靈活地修改其原始程式碼，以滿足您的特定需求。

儘管是開源軟體，您也可以選擇從 Oracle 購買商業許可證，以獲得優質支援服務。

與 Oracle Database 或 **Microsoft SQL Server** 等其他資料庫軟體相比，MySQL 相對容易掌握。

MySQL 功能多樣，可在 UNIX、Linux 和 Windows 等**各種平台上運作**。您可以將其安裝在伺服器甚至桌面上。此外，MySQL 以其**可靠性、可擴展性和速度而聞名**。

> MySQL 的正式發音是My Ess Que Ell，而不是 My Sequel。
> 不過，也可以根據自己的喜好隨意發音，這完全取決於您的個人喜好。

SQL由三個部分組成：

- 資料定義語言（DDL）包含用於定義資料庫及其物件（如表格、檢視、觸發器、預存程序等）的語句。

- 資料操作語言（DML）包含用於更新和查詢資料的語句。
- 資料控制語言 (DCL) 可讓您授予使用者存取資料庫中特定資料的權限。

安裝 MySQL 資料庫伺服器

- MySQL 下載、安裝和設定

連接到 MySQL 伺服器

1. 初次設定root密碼（MySQL初始無密碼情況）
   在Windows或Linux系統中，若MySQL初始安裝後root帳號無密碼，可以透過MySQL命令列直接設定：

```
ALTER USER 'root'@'localhost' IDENTIFIED BY '您的新密碼';
```

執行步驟：

- 以root身份登入MySQL（無密碼時直接登入）
- 輸入上述ALTER USER指令設定密碼
- 輸入 `\q` 退出MySQL

安裝 VSCode Plugin

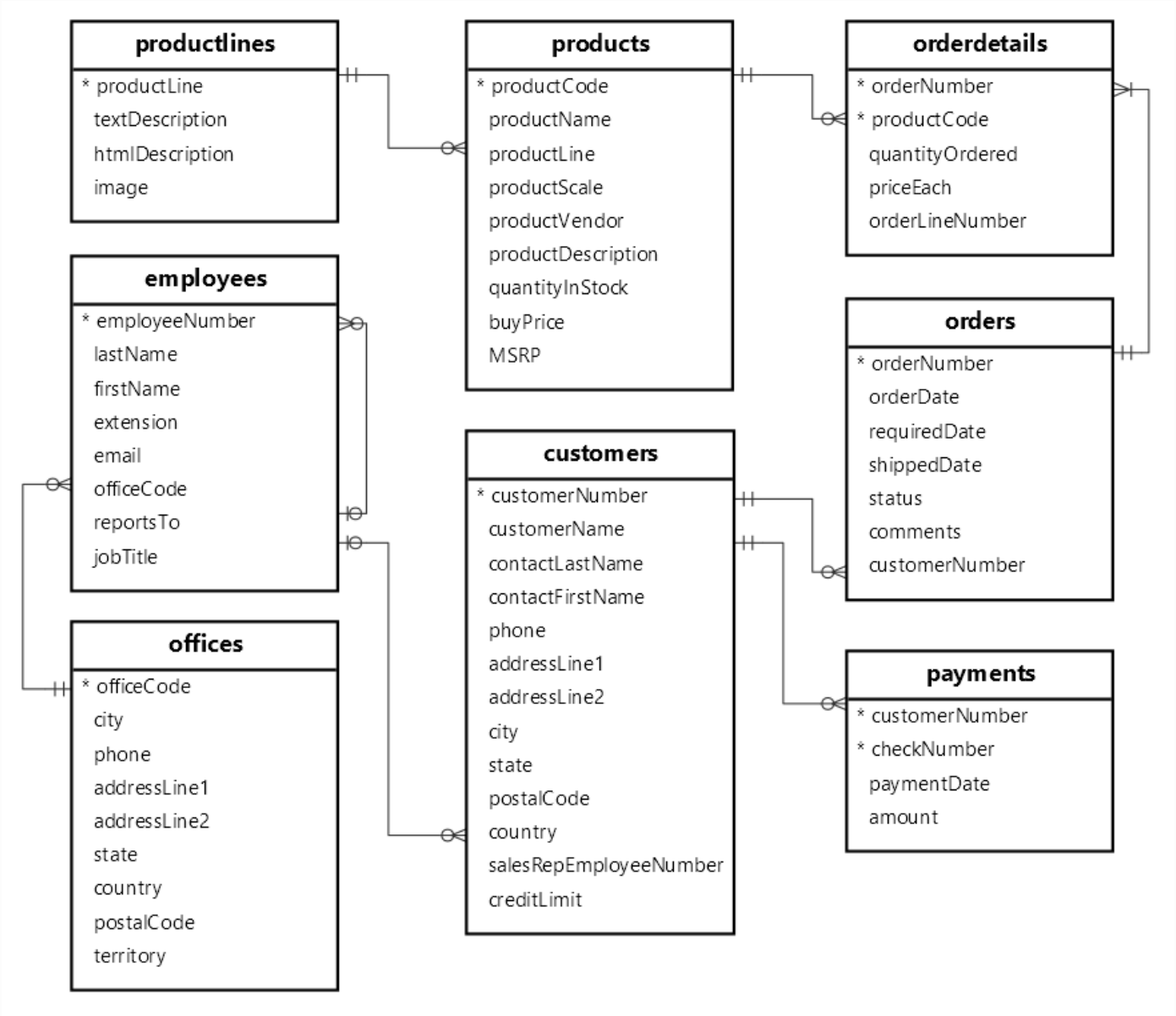
下載範例資料庫

MySQL 範例資料庫 經典汽車比例模型零售商的資料庫 由下表組成：

- 客戶 (customers)：儲存客戶的資料。
- 產品 (products)：儲存比例模型汽車的清單。
- 產品線 (productlines)：儲存產品線清單。
- 訂單 (orders)：儲存客戶下達的銷售訂單。
- 訂單明細 (orderdetails)：儲存每個銷售訂單的銷售訂單明細商品。

- 付款 (payments)：儲存客戶根據其帳戶進行的付款。
- 員工 (employees)：儲存員工資訊和組織結構，例如誰向誰報告。

辦公室 (offices)：儲存銷售辦公室資料。



查詢數據

SELECT FROM

語法：

```
SELECT select_list
FROM table_name;
```

在此語法中：

- select_list 選擇資料表欄位，多個欄位使用 `,` 隔開。
- 在 select_list 後面 FROM 指定選擇的資料表名稱。
- 分號 (;) 是可選的，它表示語句的結束。如果有兩個或多個語句，需使用分號 (;) 隔開，以便 MySQL 分別執行每個語句。

依慣例，SQL 關鍵字應大寫。但不是強制性。由於 SQL 不區分大小寫，因此 SQL 語句可以採用小寫、大寫等形式。例如SELECT：FROM

```
select select_list
from table_name;
```

SELECT FROM 範例
使用範例資料庫 employees 來進行：

- 表格 employees 有八個欄位：employeeNumber、lastName、firstName、extension、email、officeCode、reportsTo 和 jobTitle。

    1. 查詢 lastName 欄位

```
SELECT lastName
FROM employees;
```

    2. 查詢多個欄位，如 lastName、firstName、email

```
SELECT lastName, firstName, jobTitle
FROM employees;
```

    3. 查詢全部欄位

```
SELECT *
FROM employees;
```

SELECT

SELECT不引用任何表格的 MySQL 語句。
語法：

```
SELECT select_list;
```

SELECT語句執行簡單的計算：

```
SELECT 1 + 1;
```

MySQL 有許多內建函數，例如字串函數、日期函數和數學函數。可以使用 SELECT 語句來執行這些函數。
例如，使用 NOW() 函數傳回 MySQL **伺服器所在地**現在的日期和時間：

```
SELECT NOW();
```

將多個字串連接成一個字串，可以使用下列 CONCAT() 函數：

```
SELECT CONCAT('Troie',' ','Pan');
```

MySQL 使用查詢子句中指定運算式 SELECT 作為結果集的欄位名稱。若要變更結果集的欄位名稱，可以使用欄位別名：

```
SELECT expression AS column_alias;
```

```
SELECT 1+1 AS hello;
```

AS關鍵字是可選的，可以跳過它：

```
SELECT 1+1 hello;
```

```
SELECT CONCAT('Troie',' ','Pan') AS 'Full name';
```

使用別名可提高可讀性。

ORDER BY

如何使用 MySQL 子句對結果集行進行排序
語法:

```
SELECT
   select_list
FROM
   table_name
ORDER BY
   column1 [ASC|DESC],
   column2 [ASC|DESC],
   ...;
```

在此語法中，可以指定要在 **ORDER BY** 子句後面排序的一個或多個欄位。
ASC 代表升冪排序，**DESC** 代表降冪排序。使用 **ASC** 依升冪對結果進行排序，**DESC** 依降冪對結果進行排序。

1. 使用 ORDER BY 子句以一列對結果集進行排序範例

```
SELECT
  contactLastname,
  contactFirstname
FROM
  customers
ORDER BY
  contactLastname;
```

2. 使用 ORDER BY 子句以多列對結果集進行排序範例

```
SELECT
  contactLastname,
  contactFirstname
FROM
  customers
ORDER BY
  contactLastname DESC,
  contactFirstname ASC;
```

3. 使用 ORDER BY 子句按表達式對結果集進行排序範例

```
SELECT
  orderNumber,
  orderlinenumber,
  quantityOrdered * priceEach
FROM
  orderdetails
ORDER BY
  quantityOrdered * priceEach DESC;
```

加上別名

```
SELECT
  orderNumber,
  orderLineNumber,
  quantityOrdered * priceEach AS subtotal
FROM
  orderdetails
ORDER BY
  subtotal DESC;
```

透過自訂清單對資料進行排序

FIELD() 函數傳回值清單中某個值的**索引**（位置）。
語法：

```
FIELD(value, value1, value2, ...)
```

語法中：

- value：要找出其**位置**的值。
- value1, value2, ...：要與指定值進行比較的值清單。
  此函數傳回值列表中 value1、value2 等 FIELD() 的**位置**。
  如果在清單中沒找到 value，FIELD() 函數會傳回 0。

範例一:

```
SELECT FIELD('A', 'A', 'B','C');
```

範例二:
假如想根據銷售訂單(orders)的狀態依以下順序進行排序：

In Process
On Hold
Cancelled
Resolved
Disputed
Shipped

```
SELECT
  orderNumber,
  status
FROM
  orders
ORDER BY
  FIELD(
    status,
    'In Process',
    'On Hold',
    'Cancelled',
    'Resolved',
    'Disputed',
    'Shipped'
  );
```

NULL

在 MySQL 中，NULL 位於非 NULL 值之前。因此 ORDER BY 使用帶有 ASC 選項的子句時，NULLs 會優先出現在結果集中。

```
SELECT
  firstName,
  lastName,
  reportsTo
FROM
  employees
ORDER BY
  reportsTo;
```

WHERE

允許您為查詢傳回的行指定搜尋條件。

語法：

```
SELECT
    select_list
FROM
    table_name
WHERE
    search_condition;
```

`search_condition` 是指使用邏輯運算子 `AND` 、 `OR` 和 `NOT` 的一個或多個表達式的組合。
其計算結果為TRUE、FALSE或UNKNOWN。
該語句將包括結果集中 SELECT 滿足條件的任何行。

除了 `SELECT` 語句之外，您還可以使用 `WHERE` 子句指定要更新 `UPDATE` 或刪除 `DELETE` 的行。

範例：

1. 使用相等 = 運算子

   使用 WHERE 尋找所有職務為 Sales Rep 的員工：

```
SELECT
    lastname,
    firstname,
    jobtitle
FROM
    employees
WHERE
    jobtitle = 'Sales Rep';
```

2. 使用 AND 運算子

   使用 WHERE 尋找職務為 Sales Rep 且辦公室代碼為 1 的員工：

```
SELECT
    lastname,
    firstname,
    jobtitle,
    officeCode
FROM
    employees
WHERE
    jobtitle = 'Sales Rep' AND
    officeCode = 1;
```

3. 使用 OR 運算子

   尋找職務為 Sales Rep 或辦公室代碼為 1 的員工：

```
SELECT
    lastName,
    firstName,
    jobTitle,
    officeCode
FROM
    employees
WHERE
    jobtitle = 'Sales Rep' OR
    officeCode = 1
ORDER BY
    officeCode ,
    jobTitle;
```

4. 使用 WHERE 子句和 BETWEEN 運算子範例
   某個值介於值與另一個值的範圍內，則 BETWEEN 傳回：TRUE

```
expression BETWEEN low AND high
```

查詢辦公室代碼為 1 至 3 的辦公室的員工：

```
SELECT
    firstName,
    lastName,
    officeCode
FROM
    employees
WHERE
    officeCode BETWEEN 1 AND 3
ORDER BY officeCode;
```

> **TryIt**
> 查詢辦公室代碼為 2 , 4, 5, 6 的辦公室的員工：

5. 使用 LIKE 運算子
   LIKE 運算子用來評估某個值是否與指定的模式相符。
   要形成模式，可以使用 `%` 和 `_` 通配符。

- %：零或多個字元的任意字串，
  - `WHERE lastName LIKE 'a%'`：找出所有以"a"開頭的字串。
  - `WHERE lastName LIKE '%a'`：找出所有以"a"結尾的字串。
  - `WHERE lastName LIKE '%or%'`：找出任何位置包含"or"的字串。
- _：任意單一字元。
  - `WHERE lastName LIKE '_r%'`：找出第二個字元是"r"的字串。
  - `WHERE lastName LIKE 'a_%'`：找出以"a"開頭且長度至少2的字串。
  - `WHERE lastName LIKE 'a__%'`：找出以"a"開頭且長度至少3的字串。

查詢 lastName 是 'son' 結尾的員工：

```
SELECT
    firstName,
    lastName
FROM
    employees
WHERE
    lastName LIKE '%son'
ORDER BY firstName;
```

> **TryIt**
> 在表單 products 找出 productName 開頭為 196x 的所有產品

6. 使用 IN 運算子範例
   如果某個值與清單中的任意值比對，則運算子IN會傳回。TRUE

```
value IN (value1, value2,...)
```

尋找辦公室代碼為 1、2 和 3 的辦公室的員工：

```
SELECT
    firstName,
    lastName,
    officeCode
FROM
    employees
WHERE
    officeCode IN (1 , 2, 3)
ORDER BY
    officeCode;
```

> **TryIt**
> 在表單 offices 尋找位於美國(USA)和法國(France)的辦事處

7. 使用帶有 IS NULL 運算子的 MySQL WHERE 子句
   若要檢查某個值是否為 `NULL` ，請使用 `IS NULL` 運算符，而不是等號運算符（=)。該 `IS NULL` 運算子傳回 TRUE 值則為 NULL。

```
value IS NULL
```

> 在資料庫中 NULL 是一個標記，表示值缺失或未知。 `NULL` 不等於數字 0 或空字串。

取得 reportsTo 列 IS NULL 中值為 NULL 的行：

```
SELECT
    lastName,
    firstName,
    reportsTo
FROM
    employees
WHERE
    reportsTo IS NULL;
```

> **TryIt**
> 在表單 orders 找出 shippedDate 為 NULL 的所有訂單，並以 orderDate 排序

8. 比較運算子

運算子	描述
=	等於。幾乎可以將其與任何資料類型一起使用。
<> 或 !=	不等於
<	小於。通常將其用於數字和日期/時間資料類型。
>	大於。
<=	小於或等於
>=	大於或等於

尋找所有不屬於 Sales Rep 的員工：

```
SELECT
    lastname,
    firstname,
    jobtitle
FROM
    employees
WHERE
    jobtitle <> 'Sales Rep';
```

> **TryIt**
> 在表單 orders 找出 status 不等於 Shipped 及 Cancelled 的所有訂單，並以 orderDate 排序

> **TryIt**
> 在表單 products 找出欄位為 productName, productDescription, quantityInStock, buyPrice
> 以及 quantityInStock 介於 1000 到 2000 的所有產品

SELECT DISTINCT

從表格查詢資料時，可能會出現重複行。若要刪除這些重複行，請在 SELECT 語句中使用 DISTINCT 子句。

語法：

```
SELECT DISTINCT
    select_list
FROM
    table_name
WHERE
    search_condition
ORDER BY
    sort_expression;
```

範例：
從表格 employees 中列出所有的 lastname：

```
SELECT
    lastname
FROM
    employees
ORDER BY
    lastname;
```

每個 lastname 只留下一個（不重複）

```
SELECT
    DISTINCT lastname
FROM
    employees
ORDER BY
    lastname;
```

> **TryIt**
> 在表單 products 有那些產品線，每個產品線個有哪些縮放比 productLine

DISTINCT 具有多個欄位

從 customers 表中取得城市和州的不重複組合：

```
SELECT DISTINCT
    state, city
FROM
    customers
WHERE
    state IS NOT NULL
ORDER BY
    state,
    city;
```

> **TryIt**
> 在表單 products 有那些產品線，每個產品線個有哪些縮放比 productScale

AND

`AND` 是一個邏輯運算符，它組合兩個或多個布林表達式並傳回 1、0 或 NULL：

`A AND B`

在這個表達式中，A 和 B 稱為運算元。它們可以是文字值，也可以是表達式。

如果 A 和 B 都非零且不是 NULL，則邏輯與運算子傳回 1。如果其中一個運算元為零，則傳回 0；否則，傳回 NULL。

範例：
尋找位於美國加州 (CA) 的客戶：

```
SELECT
    customername,
    country,
    state
FROM
    customers
WHERE
    country = 'USA' AND
    state = 'CA';
```

> **TryIt**
> 在表單 products 找出產品線productLine 為 Trucks 且價格低於 50 的產品

OR

> **TryIt**
> 在表單 customers 找出於美國(country)或法國且信用額度(creditLimit)大於 100,000 的客戶(customername)。
> 提示 `where (A or B) and C`

```
SELECT
  customername,
  country,
  creditLimit
FROM
  customers
WHERE(country = 'USA' OR country = 'France')
    AND creditlimit > 100000;
```

NOT IN

尋找不在 France 和 USA 的辦公室

```
SELECT
    officeCode,
    city,
    phone
FROM
    offices
WHERE
    country NOT IN ('USA' , 'France')
ORDER BY
    city;
```

BETWEEN

傳回所需日期在 2003 年 1 月 1 日至 2003 年 1 月 31 日之間的訂單

```
SELECT
   orderNumber,
   requiredDate,
   status
FROM
   orders
WHERE
   requireddate BETWEEN
     CAST('2003-01-01' AS DATE) AND
     CAST('2003-01-31' AS DATE);
```

使用 CAST() 將文字字串轉換 '2003-01-01' 為一個 DATE 值：

```
CAST('2003-01-01' AS DATE)
```

LIMIT

限制要傳回的行數。接受一個或兩個參數。兩個參數的值都必須為零或正整數。

語法：

```
SELECT
    select_list
FROM
    table_name
LIMIT [offset,] row_count;
```

在此語法中：

- 指定 offset 要傳回的第一行的偏移量。offset 第一行的偏移量為 0，而不是 1。
- 指定 row_count 要傳回的最大行數。

範例：

1. 取得信用最高的前五名客戶

```
SELECT
    customerNumber,
    customerName,
    creditLimit
FROM
    customers
ORDER BY creditLimit DESC
LIMIT 5;
```

2. 使用 LIMIT 進行分頁
   首先取得總客戶數

```
SELECT
    COUNT(*)
FROM
    customers;
```

假設每頁有 20 行；若要顯示 122 位客戶，則需要 7 頁。最後一頁僅包含兩行。

```
SELECT
    customerNumber,
    customerName
FROM
    customers
ORDER BY customerName
LIMIT 0, 20;
```

接著取得後 20 筆

```
SELECT
    customerNumber,
    customerName
FROM
    customers
ORDER BY customerName
LIMIT 20, 20;
```

建立資料庫

CREATE DATABASE 語句簡介
若要在 MySQL 中建立新資料庫，請使用該CREATE DATABASE語句。以下說明了該CREATE DATABASE語句的基本語法：

```
CREATE DATABASE [IF NOT EXISTS] database_name
[CHARACTER SET charset_name]
[COLLATE collation_name];
```

在此語法中：

1. 在關鍵字 `CREATE DATABASE` 後面指定資料庫的名稱。資料庫名稱在 MySQL 伺服器實例中必須是唯一的。如果嘗試建立具有現有名稱的資料庫，MySQL 將彈出錯誤。
2. `IF NOT EXISTS` 如果資料庫不存在，則使用有條件建立資料庫的選項。
3. 指定新資料庫的字元集和排序規則 。如果跳過 CHARACTER SET and COLLATE 子句，MySQL 將使用新資料庫的預設字元集和排序規則。

範例

```
CREATE DATABASE HELLO
    DEFAULT CHARACTER SET = 'utf8mb4'
    COLLATE utf8mb4_0900_ai_ci;
```

COLLATE 是用來定義排序規則（collation），決定了：

- 字符比較的方式
- 排序的順序
- 大小寫敏感性

utf8mb4_0900_ai_ci 的特性：

-
  - `utf8mb4`：對應的字符集
- ai：Accent Insensitive（不區分重音符號）
- ci：Case Insensitive（不區分大小寫）
- 基於 Unicode 9.0 標準
- 對繁體中文排序最準確

- 支援完整的中文字符排序

常見的 utf8mb4 排序規則

1. **utf8mb4_general_ci**
    - 不區分大小寫
    - 速度快，但某些語言排序可能不夠精確
2. **utf8mb4_unicode_ci**
    - 不區分大小寫
    - 更精確的 Unicode 排序，但速度稍慢
3. **utf8mb4_bin**
    - 區分大小寫
    - 按字節值進行比較
4. **utf8mb4_0900_ai_ci**（MySQL 8.0 預設）
    - 不區分大小寫和重音符號
    - 基於 Unicode 9.0 標準

> 如果需要更精確的排序（特別是多語言環境），可以考慮使用 `utf8mb4_unicode_ci`。

資料庫列表

查詢已完成資料庫列表

語法：

```
SHOW DATABASES;
```

刪除資料庫

刪除資料庫中的所有表並**永久刪除資料庫**。因此，使用此語句時需要非常小心。

語法：

```
DROP DATABASE [IF EXISTS] database_name;
```

如果刪除不存在的資料庫，MySQL 將彈出錯誤。

為了防止刪除不存在的資料庫時發生錯誤，可以使用該 `IF EXISTS` 選項。在這種情況下，MySQL 將終止該語句而不發出任何錯誤。

`DROP DATABASE` 會傳回刪除資料表的數量。

建立資料表

使用 `CREATE TABLE` 語句在資料庫中建立新表。

語法：

```
CREATE TABLE [IF NOT EXISTS] table_name(
   column1 datatype constraints,
   column2 datatype constraints,
   ...
) ENGINE=storage_engine;
```

在此語法中：

- table_name：要建立的資料表名稱。
- column1、、column2等：表中欄位的名稱。
- datatype：各欄位的數據型別如 INT、VARCHAR、DATE... 等

- constraints：這些是可選約束，例如 NOT NULL、UNIQUE 和 PRIMARY KEYFOREIGN KEY。
  如果要建立的資料表名稱於資料庫中已存在相同名稱，則會出錯。為了避免此錯誤，可以使用該 `IF NOT EXISTS` 選項。

如果沒有明確指定儲存引擎，MySQL 將使用預設儲存引擎 InnoDB。

> 從 MySQL 5.5 版開始，InnoDB 成為預設儲存引擎。 InnoDB 儲存引擎提供了關聯式資料庫管理系統的多項優勢，包括 ACID 事務支援、參考完整性和崩潰復原。在早期版本中，MySQL 使用 MyISAM 作為預設儲存引擎。

範例：

1. 建立一個名為 tasks 的資料表：

```
CREATE TABLE tasks (
    id INT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    start_date DATE,
    due_date DATE
);
```

在此語法中：

- id 是一個 INT 型別，當成主鍵。
- title 是 VARCHAR 型別，不能可以是 NULL。
- start_date, due_date 是 DATE 型別，可以是 NULL 。

檢視資料表

查詢資料庫內所有資料表。
語法：

```
SHOW TABLES;
```

查詢資料表詳細欄位形態及相關資訊。
語法：

```
DESCRIBE TALBE_NAME;
```

- TALBE_NAME 輸入欲查詢資料表名稱。

資料類型

字串類型 (String Data Types)

資料類型	說明	範例
`CHAR(size)`	固定長度字串，長度為 size 個字元（最多 255），**不足部分會補空白字元**	`CHAR(10)`：固定取 10 字元長度
`VARCHAR(size)`	可變長度字串，最多 size 個字元（最大 65535），**依實際字元長度存儲**	`VARCHAR(50)`：最多 50 字元變長字串
`TEXT`	大型字串，最多 65,535 字元	商品描述欄位常用
`TINYTEXT`	小型字串，最多 255 字元	簡短備註
`BLOB`	二進位大物件，可儲存最多 65,535 bytes	儲存圖片、檔案等二進位資料
`ENUM(val1,val2,...)`	列舉字串欄位，只能從指定列表中選擇一個值	`ENUM('男','女')`：性別欄

數值類型 (Numeric Data Types)

資料類型	說明	範例
`TINYINT`	非常小的整數，範圍：-128 到 127（有號）或 0 到 255（無號）	`TINYINT UNSIGNED`：0~255
`SMALLINT`	小整數，範圍：-32,768 到 32,767（有號）或 0 到 65,535（無號）	`SMALLINT`
`INT` 或 `INTEGER`	中等大小整數，範圍：-2,147,483,648 到 2,147,483,647（有號）	`INT UNSIGNED`
`BIGINT`	大整數，範圍：-9,223,372,036,854,775,808 到 9,223,372,036,854,775,807（有號）	用於存超大數字
`FLOAT`	浮點數，近似值，適合存小數。 MySQL 8.0 以後建議只用 `FLOAT(p)` 格式	`FLOAT`
`DOUBLE`	雙精度浮點數，精確度更高	`DOUBLE`
`DECIMAL(size,d)`	精確定點數，size 是總位數，d 是小數點位數	`DECIMAL(10,2)`：最多10位，其中2位為小數
`BIT(size)`	位元欄位，可設定儲存幾位元，size 範圍 1 到 64	`BIT(8)`
`BOOLEAN`	布林值，等同於 `TINYINT(1)`，0 是假，非0是真	`BOOLEAN`

日期與時間類型 (Date and Time Data Types)

資料類型	說明	範例
`DATE`	日期，格式：`YYYY-MM-DD`，範圍：1000-01-01 到 9999-12-31	`2025-08-11`
`DATETIME`	日期和時間，格式：`YYYY-MM-DD HH:MM:SS`，範圍：1000-01-01 00:00:00 到 9999-12-31 23:59:59	`2025-08-11 22:30:00`
`TIMESTAMP`	時間戳記，儲存從 Unix 紀元(1970-01-01)起的秒數，範圍：1970-01-01 到 2038-01-19	自動更新欄位常用
`TIME`	時間，格式：`HH:MM:SS`，可儲存負時間和值範圍：-838:59:59 到 838:59:59	`23:59:59`
`YEAR`	年份，四位數格式，範圍：1901 到 2155	`2025`

JSON 類型

資料類型	說明	範例
`JSON`	用於儲存結構化的 JSON 格式資料，可直接用 SQL 操作 JSON 物件與陣列	`{"name": "John", "age": 30}`

範例表格建立

```
CREATE TABLE `products` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(50) NOT NULL,
  `price` DECIMAL(10,2) NOT NULL,
  `description` TEXT,
  `image` BLOB,
  `created_at` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `updated_at` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

範例中：

- `id` 使用無號整數，並自動遞增 `AUTO_INCREMENT` 。
- `name` 使用可變長字串。
- `price` 使用定點數，精確到小數點後兩位。
- `description` 使用長文本字串。
- `image` 使用二進位大型物件儲存。
  將 image 欄位改為 image_url，並將其類型設為 VARCHAR(255)，是目前業界公認的最佳作法。
- `created_at` 與 `updated_at` 管理時間戳記，自動插入與更新。

> **AUTO_INCREMENT**
>
> - 使用 `AUTO_INCREMENT` 屬性自動為資料列產生唯一的整數值。
> - 通常主鍵使用 `AUTO_INCREMENT` 來確保每一行都有唯一的識別碼。
> - 新增一筆資料時 `AUTO_INCREMENT` ，不需要指定該列的值，MySQL 會自動產生該值。

取得最後一個自動遞加值

若要取得 `AUTO_INCREMENT` 最新插入(最後一筆)產生的值，請使用下列 `LAST_INSERT_ID()` 函數：

```
SELECT LAST_INSERT_ID();
```

資料表更名

重新命名現有的資料表以更好地適應新情況。 MySQL 提供了一個非常有用的語句，可用來重新命名一個或多個表。

語法：

```
RENAME TABLE table_name
TO new_table_name;
```

在此語法中：

- table_name：要重新命名資料表的名稱。
- new_table_name：新的資料表名稱。

執行 `RENAME TABLE` 時，要確保沒有活動事務或鎖定的表。

新增資料欄位

語法：

```
ALTER TABLE table_name
ADD COLUMN new_column_name data_type
[FIRST | AFTER existing_column];
```

在此語法中：

1. 提供要在 `ALTER TABLE` 子句後面是新增欄位的資料表。
2. 在 `ADD COLUMN` 子句後面定義新增欄位及其屬性。
3. 新增欄位在資料表中的位置。

在表中新增欄位時，可以指定其在表中的位置。如果希望新欄位位於表格的第一欄，可以使用關鍵字。

或者使用 `AFTER existing_colum` 子句指定要在現有欄位後面新增欄位。

如果沒有明確指定新增欄位的位置，語句將自動將其新增於表格中的最後一列。

若要同時在表中新增兩個欄位或更多欄位，可以使用 `ADD COLUMN` 下列多個子句：

```
ALTER TABLE table_name
ADD [COLUMN] new_column_name data_type [FIRST|AFTER
existing_column],
ADD [COLUMN] new_column_name data_type [FIRST|AFTER
existing_column],
...;
```

練習：

建立一個新資料表

```
CREATE TABLE vendors (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255)
);
```

1. 新增一個 `phone` 欄位

```
ALTER TABLE vendors
ADD COLUMN phone VARCHAR(15) AFTER name;
```

2. 確認新增欄位 `DESC vendors;`
3. 同時新增 email 及 hourly_rate 兩個欄位均為 `NOT NULL`

```
ALTER TABLE vendors
ADD COLUMN email VARCHAR(100) NOT NULL,
ADD COLUMN hourly_rate decimal(10,2) NOT NULL;
```

刪除資料欄位

從表格中刪除一個或多個資料欄位。

語法：

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

在此語法中：

1. 在 `ALTER TABLE` 關鍵字後面指定包含要刪除的資料表的名稱。
2. 指定要在 `DROP COLUMN` 子句中刪除欄位的名稱。

也可以使用較短的語句，如下：

```
ALTER TABLE table_name
DROP column_name;
```

如果要從表格中刪除多個欄位，使用下列語法：

```
ALTER TABLE table_name
DROP COLUMN column_name_1,
DROP COLUMN column_name_2,
...;
```

- 從表中刪除一個欄位會導致所有引用該欄位的資料庫物件（例如預存程序、檢視和觸發器）失效。例如，您可能有一個引用該欄位的預存程序。當您刪除該欄位時，該預存程序將失效。若要修復此問題，您必須手動變更預存程序的程式碼。
- 依賴刪除的欄位的其他應用程式的程式碼也必須更改，這需要時間和精力。
- 從大表中刪除一欄位可能會在刪除期間影響資料庫的效能。

練習：
建立一個新表單

```
CREATE TABLE posts (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    excerpt VARCHAR(400),
    content TEXT,
    created_at DATETIME,
    updated_at DATETIME
);
```

1. 刪除 `excerpt` 欄位。

```
ALTER TABLE posts
DROP COLUMN excerpt;
```

2. 確認完成刪除。
3. 同時刪除 `created_at` 和 `updated_at` 欄位。

刪除資料表

從資料庫中刪除表格。

```
DROP [TEMPORARY] TABLE [IF EXISTS] table_name [,
table_name] ...
[RESTRICT | CASCADE]
```

從資料庫中永久刪除一個資料表及其資料。在 MySQL 中，可以使用單一 `DROP TABLE` 刪除多個表，每個表之間以逗號 (,) 分隔。

此 `TEMPORARY` 選項僅允許您刪除臨時表。它確保您不會意外刪除非臨時表。

`IF EXISTS` 選項僅在表存在時才有條件地刪除該表。如果刪除不存在的表，MySQL 將產生一條註釋，您可以使用該 `SHOW WARNINGS` 語句檢索該註釋。

`RESTRICT` 和 `CASCADE` 選項保留用於 MySQL 的未來版本。

要執行 `DROP TABLE` 您必須擁有 `DROP` 要刪除的表的權限。

練習：

1. 刪除 posts 資料表

```
DROP TABLE posts;
```

2. 刪除多個表

   建立兩個資料表 CarAccessories 和 CarGadgets。

```
CREATE TABLE CarAccessories (
    id INT AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    price DEC(10 , 2 ) NOT NULL,
    PRIMARY KEY(id)
);

CREATE TABLE CarGadgets (
    id INT AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    price DEC(10 , 2 ) NOT NULL,
    PRIMARY KEY(id)
);
```

```
DROP TABLE CarAccessories, CarGadgets;
```

3. 刪除不存在的表

```
Error Code: 1051. Unknown table 'classicmodels.aliens'
```

正確做法

```
DROP TABLE IF EXISTS aliens;
```

建立臨時表

臨時表（Temporary Tables）是一種特殊的表格，它只存在於**當前的資料庫連線（session）中**。當這個連線結束時，臨時表會被自動刪除。

臨時表主要用於以下情況：

1. **儲存中間結果：** 當一個複雜的查詢需要分多個步驟執行，或是需要對查詢結果進行多次操作時，你可以將中間結果儲存在臨時表中，然後再對其進行後續處理，這樣可以讓查詢邏輯更清晰。
2. **效能優化：** 在處理大量資料時，與其在一個龐大的查詢中進行多個 `JOIN` 和子查詢，不如將部分結果暫存到臨時表中，然後對這個較小的臨時表進行操作，這通常可以提高查詢效能。
3. **減少重複計算：** 如果你需要在一個會話中多次使用相同的子查詢結果，將其儲存在臨時表中可以避免重複計算，節省時間。

如何創建和使用臨時表？

臨時表的語法與創建普通表格非常相似，只需在 `CREATE TABLE` 後面加上 `TEMPORARY` 關鍵字。

1. 創建臨時表

可以用兩種方式創建：

- **從頭創建：**

```
CREATE TEMPORARY TABLE temp_products (
  id INT,
  name VARCHAR(255),
  price DECIMAL(10, 2)
);
```

- **從現有查詢結果創建：** 這是更常見和實用的方式，可以同時創建表格並載入資料。

```
CREATE TEMPORARY TABLE popular_products AS
SELECT product_id, COUNT(*) AS total_sales
FROM sales
GROUP BY product_id
ORDER BY total_sales DESC
LIMIT 10;
```

這個語法會創建一個名為 `popular_products` 的臨時表，並將查詢結果（前十名熱銷商品）存入其中。

2. 使用臨時表

使用臨時表的方式與普通表格完全一樣，你可以進行 `SELECT`、`INSERT`、`UPDATE`、`DELETE` 等操作。

```
SELECT * FROM popular_products;

INSERT INTO temp_products VALUES (1, '新產品', 150.00);

UPDATE temp_products SET price = 200.00 WHERE id = 1;
```

3. 刪除臨時表

當連線結束時，臨時表會被自動刪除，你也可以手動刪除它：

```
DROP TEMPORARY TABLE temp_products;
```

注意事項

- **名稱範圍：** 臨時表的名稱是**會話（session）專屬的**。這意味著兩個不同的使用者可以同時創建同名的臨時表，而不會互相影響。
- **不支援 `SHOW TABLES`：** 在 MySQL 8.0.33 之前的版本，`SHOW TABLES` 命令不會列出臨時表。但從 8.0.33 版本開始，在當前會話中執行 `SHOW TABLES` 會顯示臨時表。
- **權限要求：** 創建臨時表需要有 `CREATE TEMPORARY TABLES` 的權限。
- **交易（Transactions）：** 臨時表的創建與刪除**不受交易的影響**。即使在 `ROLLBACK` 後，臨時表仍然存在。

自動化欄位 Generated Columns

透過表達式或其他欄位計算的資料。

在 `CREATE TABLE` 語句中指定表格欄位，然後使用 `INSERT` 、 `UPDATE` 和 `DELETE` 語句直接修改表格欄位中的資料。

首先建立一個資料表：

```
CREATE TABLE contacts (
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL
);
```

如果要取得聯絡人的全名需使用 `CONCAT()` 函數：

```
SELECT
    id,
    CONCAT(first_name, ' ', last_name),
    email
FROM
    contacts;
```

透過 `Generated Columns` 重新建立表：

```
DROP TABLE IF EXISTS contacts;

CREATE TABLE contacts (
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    fullname varchar(101) GENERATED ALWAYS AS
(CONCAT(first_name,' ',last_name)),
    email VARCHAR(100) NOT NULL
);
```

`GENERATED ALWAYS as (expression)` 是建立生成欄位的語法。

```
INSERT INTO contacts(first_name,last_name, email)
VALUES('john','doe','john.doe@mysqltutorial.org');
程式碼語言： SQL（結構化查詢語言） （sql ）
現在，您可以從表格中查詢資料contacts。
```

```
-- 插入第一筆資料：環保木質餐桌
INSERT INTO `products` (`name`, `price`, `description`,
`image`) VALUES
('環保木質餐桌', 7999.00, '採用永續林業木材製造，簡約風格適合現代家
庭，表面經特殊處理，耐磨且易於清潔。', NULL);

-- 插入第二筆資料：多功能收納櫃
INSERT INTO `products` (`name`, `price`, `description`,
`image`) VALUES
('多功能收納櫃', 3500.50, '組裝簡單，多層次收納空間，有效利用居家角
落，為您的生活帶來整潔與便利。', NULL);

-- 插入第三筆資料：人體工學辦公椅
INSERT INTO `products` (`name`, `price`, `description`,
`image`) VALUES
('人體工學辦公椅', 2280.00, '符合人體脊椎曲線設計，提供全方位支撐，
久坐也不易疲勞，是您辦公與學習的最佳夥伴。', NULL);
```

將「環保木質餐桌」的價格從 7999.00 改為 8500.00，並新增一段描述。

```
UPDATE `products`
SET
  `price` = 8500.00,
  `description` = '採用永續林業木材製造，簡約風格適合現代家庭，表面
經特殊處理，耐磨且易於清潔。此商品現正特價中！'
WHERE
  `name` = '環保木質餐桌';
```

插入 INSERT

向表中插入一行或多行紀錄。

語法：

```
INSERT INTO table_name(column1, column2,...)
VALUES (value1, value2,...);
```

- 使用 `INSERT` 時，需要確保列數與值數相符。
- 需要指定列的位置 `column` 與其對應值 `value` 的位置精確對應。

使用單一語句將多行紀錄插入表中：

```

INSERT INTO table(column1, column2,...)
VALUES
  (value1, value2,...),
  (value1, value2,...),
  ...
  (value1, value2,...);
```

以逗號分隔 VALUES。

建立一個 tasks 新表格：

```
CREATE TABLE tasks (
  task_id INT AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(255) NOT NULL,
  start_date DATE,
  due_date DATE,
  priority TINYINT NOT NULL DEFAULT 3,
  description TEXT
);
```

> 練習：新增一筆紀錄 title：Learn MySQL INSERT Statement，priority：1

```
INSERT INTO tasks(title, priority)
VALUES('Learn MySQL INSERT Statement', 1);
```

2. 使用 DEFAULT 值新增紀錄

```
INSERT INTO tasks(title, priority)
VALUES('Understanding DEFAULT keyword', DEFAULT);
```

3. 插入日期格式

若要將文字日期值插入列，請使用下列格式：

'YYYY-MM-DD'

- YYYY 代表四位數的年份，例如 2018。
- MM 表示兩位數的月份，例如 01、02 和 12。
- DD 代表兩位數的日期，例如 01、02、30。

```
INSERT INTO tasks(title, start_date, due_date)
VALUES ('Insert date into table', '2018-01-09', '2018-09-
15');
```

4. 使用表達式VALUES。

```
INSERT INTO tasks(title, start_date, due_date)
VALUES
  (
    'Use current date for the task',
    CURRENT_DATE(),
    CURRENT_DATE()
  );
```

> CURRENT_DATE()函數，回傳當前系統日期。

> - 功能： 擷取執行此函數時的伺服器日期。
> - 回傳值： 回傳一個 DATE 類型的值，格式為 'YYYY-MM-DD'。

5. 插入多行紀錄

```
INSERT INTO tasks(title, priority)
VALUES
  ('My first task', 1),
  ('It is the second task',2),
  ('This is the third task of the week',3);
```

插入選擇

使用 SELECT 的結果作為 INSERT 的資料來源。

```
INSERT INTO table_name(column_list)
SELECT
   select_list
FROM
   another_table
WHERE
   condition;
```

**注意，select_list 的列數與 column_list 必須相等。

建立一個 suppliers 新表：

```
CREATE TABLE suppliers (
    supplierNumber INT AUTO_INCREMENT,
    supplierName VARCHAR(50) NOT NULL,
    phone VARCHAR(50),
    addressLine1 VARCHAR(50),
    addressLine2 VARCHAR(50),
    city VARCHAR(50),
    state VARCHAR(50),
    postalCode VARCHAR(50),
    country VARCHAR(50),
    customerNumber INT,
    PRIMARY KEY (supplierNumber)
);
```

查詢尋找位於 California, USA 的所有客戶：

```
SELECT
    customerNumber,
    customerName,
    phone,
    addressLine1,
    addressLine2,
    city,
    state,
    postalCode,
    country
FROM
    customers
WHERE
    country = 'USA' AND
    state = 'CA';
```

使用以下語法將位於 customer 表中的 California，USA 客戶 INSERT INTO SELECT 插入 suppliers 表 ：

```
INSERT INTO suppliers (
    supplierName,
```

```
    phone,
    addressLine1,
    addressLine2,
    city,
    state,
    postalCode,
    country,
    customerNumber
)
SELECT
    customerName,
    phone,
    addressLine1,
    addressLine2,
    city,
    state ,
    postalCode,
    country,
    customerNumber
FROM
    customers
WHERE
    country = 'USA' AND
    state = 'CA';
```

在 VALUES 中使用 SELECT

建立一個 stats 新表：

```
CREATE TABLE stats (
    totalProduct INT,
    totalCustomer INT,
    totalOrder INT
);
```

使用 INSERT 插入來自 SELECT 的值：

```
INSERT INTO stats(totalProduct, totalCustomer, totalOrder)
VALUES(
  (SELECT COUNT(*) FROM products),
  (SELECT COUNT(*) FROM customers),
  (SELECT COUNT(*) FROM orders)
);
```

INSERT IGNORE

在新增多筆紀錄時，如果處理過程中出現錯誤，MySQL 將終止並傳回錯誤。因此，資料表保持不變，不會插入任何行。

`INSERT IGNORE` 可以忽略包含無效資料的行（否則會觸發錯誤），僅插入有效資料的行。

語法：

```
INSERT IGNORE INTO table(column_list)
VALUES(value_list),
      (value_list),
      ...;
```

範例：
建立 subscribers 新表。

```
CREATE TABLE subscribers (
    id INT PRIMARY KEY AUTO_INCREMENT,
    email VARCHAR(130) NOT NULL UNIQUE
);
```

`UNIQUE` 約束確保 `email` 列中不存在重複的電子郵件。

在 subscribers 中新增一筆紀錄：

```
INSERT INTO subscribers(email)
VALUES('john.doe@gmail.com');
```

接著再將兩筆紀錄插入 subscribers ：

```
INSERT INTO subscribers(email)
VALUES('john.doe@gmail.com'),
      ('jane.smith@ibm.com');
```

MySQL 回傳一個錯誤。

```
Error: (conn:58, no: 1062, SQLState: 23000) Duplicate entry
'john.doe@gmail.com' for key 'email' sql: INSERT INTO
subscribers(email) VALUES('john.doe@gmail.com'),
('jane.smith@ibm.com')
```

`john.doe@gmail.com` 違反了 `UNIQUE` 約束。

如果改用INSERT IGNORE語句。

```
INSERT IGNORE INTO subscribers(email)
VALUES('john.doe@gmail.com'),
      ('jane.smith@ibm.com');
```

MySQL 傳回一則訊息，表示一筆紀錄已插入，另一筆被忽略。

使用 `SHOW WARNINGS` 可查詢警告訊息：

```
SHOW WARNINGS;
```

使用該 `INSERT IGNORE` 時，如果發生錯誤，MySQL 不會發出錯誤，而是發出警告。

INSERT IGNORE 和 STRICT 模式

當開啟嚴格模式時，INSERT 如果嘗試在表中插入無效值，MySQL 將傳回錯誤並中止語句。

但是，如果使用 INSERT IGNORE 語句，MySQL 將發出警告而不是錯誤。此外，它會在將值加到表之前嘗試調整值以使其有效。

範例：

1. 建立一個名為 tokens 的新表 ：

```
CREATE TABLE tokens (
    s VARCHAR(6)
);
```

在此表中，s 只接受長度小於或等於六的字串。

2. 向表中插入一個長度為七的字串tokens。

```
INSERT INTO tokens VALUES('abcdefg');
```

由於嚴格模式處於開啟狀態，MySQL 發出以下錯誤。

```
Error: (conn:61, no: 1146, SQLState: 42S02) Table
'hello.tokens' doesn't exist sql: INSERT INTO tokens
VALUES('abcdefg')
```

3. 使用INSERT IGNORE語句插入相同的字串。

```
INSERT IGNORE INTO tokens VALUES('abcdefg');
```

MySQL 將資料插入 tokens 表之前截斷了資料。此外，還會發出警告。

插入日期時間

如何將DATETIME值插入 MySQL 資料庫的表中。

如要將資料插入 DATETIME 欄位，需要確保日期時間值符合 'YYYY-MM-DD HH:MM:SS' 格式。

如果有不同格式的日期時間值，則**需要對其進行格式化**以符合 'YYYY-MM-DD HH:MM:SS' 。

範例：

1. 建立一個 events 的表：

```
CREATE TABLE events(
    id INT AUTO_INCREMENT PRIMARY KEY,
    event_name VARCHAR(255) NOT NULL,
    event_time DATETIME NOT NULL
);
```

- event_name：此欄位儲存事件的名稱。
- event_time：此欄位是 DATETIME 的資料類型。

2. 在 events 表中插入新行並使用日期時間格式 'YYYY-MM-DD HH:MM:SS'：

```
INSERT INTO events(event_name, event_time)
VALUES('MySQL tutorial livestream', '2023-10-28 19:30:35');
```

3. 從 events 表中查詢資料：

插入目前日期時間

使用該NOW()函數作為日期時間值。例如：

```
INSERT INTO events(event_name, event_time)
VALUES('MySQL Workshop', NOW());
```

插入日期時間字串

如果要在 DATETIME 欄位中插入**日期時間字串**，則需要使用 `STR_TO_DATE()` 函數將其轉換為預期格式。

```
INSERT INTO events (event_name, event_time)
VALUES ('MySQL Party', STR_TO_DATE('10/28/2023 20:00:00',
'%m/%d/%Y %H:%i:%s'));
```

STR_TO_DATE()

將**字串**（STRING）按照指定的格式轉換成 **日期**（DATE）、**時間**（TIME）或**日期時間**（DATETIME）值。

語法

`STR_TO_DATE()` 語法如下：

```
STR_TO_DATE(str, format)
```

- str：你想要轉換的字串，例如 '2024-05-20' 或 'May 20, 2024'。
- format：一個格式字串，用來告訴 MySQL 如何解析 str。這個格式字串由特定的**格式代號**（Format Specifiers）組成。

常見的格式代號

以下是一些常用且重要的格式代號：

代號	說明	範例
%Y	四位數的年份	2024
%y	兩位數的年份	24
%m	兩位數的月份 (01-12)	05
%c	數字月份 (1-12)	5
%M	月份完整名稱	May
%b	月份縮寫名稱	May
%d	兩位數的日期 (01-31)	20
%e	數字日期 (1-31)	20
%H	兩位數的小時 (00-23)	14
%h	兩位數的小時 (01-12)	02
%i	兩位數的分鐘 (00-59)	30
%s	兩位數的秒鐘 (00-59)	05

使用時機

STR_TO_DATE() 函數在以下情況非常有用：

- **資料匯入：** 當你從外部來源（如檔案、API）取得的日期或時間資料是字串格式，且格式不固定時，需要轉換後才能存入資料庫的 DATE 或 DATETIME 欄位。
- **資料清洗：** 處理資料庫中格式不統一的日期字串。
- **動態查詢：** 在 SQL 查詢中，根據使用者輸入的不同日期字串格式進行動態轉換。

重要提醒： 如果你的字串格式不符合 `STR_TO_DATE()` 的 format 參數，它會回傳 NULL。因此，確保格式字串與你的輸入字串精確匹配非常重要。

更新 UPDATE

更改單筆或多筆紀錄中一筆或多筆紀錄的值。

語法：

```
UPDATE [LOW_PRIORITY] [IGNORE] table_name
SET
    column_name1 = expr1,
    column_name2 = expr2,
    ...
[WHERE
    condition];
```

1. 在關鍵字 UPDATE 後面指定要更新資料的資料表。
2. 指定要更新的欄位及其新值。若要更新多列的值，可以使用逗號分隔的賦值列表，並在每列的賦值中以文字值、表達式或子查詢以 SET 形式提供值。
3. 使用子句 WHERE 條件指定要更新的欄位 。WHERE 是可選的，如果省略，UPDATE語句將修改表格中的所有欄位。
   請注意， ** WHERE 非常重要**，切勿忘記。有時，可能只想更新一筆紀錄，忘記加上 WHERE，它會幫你更新所有的紀錄。

UPDATE 支援兩種修飾符。

1. LOW_PRIORITY 修飾詞指示 UPDATE 延遲更新，直到沒有連接從表中讀取資料。LOW_PRIORITY 修飾符僅對使用表級鎖定的儲存引擎有效，例如 MyISAM、MERGE 和 MEMORY。
2. IGNORE 修飾詞使 UPDATE 即使發生錯誤也能繼續更新。導致諸如重複鍵衝突之類的錯誤的行不會被更新。

範例：

1. 開啟範例資料表

```
USE DATABASE;
```

將 employees 表單的 Mary Patterson 的 email 更新為新的電子郵件 mary.patterso@classicmodelcars.com。

```
SELECT
    firstname,
    lastname,
    email
FROM
    employees
WHERE
    employeeNumber = 1056;
```

> 練習：使用 Mary Patterson 找出該筆紀錄

```
SELECT
    firstname,
    lastname,
    email
FROM
    employees
WHERE
    firstname = 'Mary'
    AND lastname = 'Patterson';
```

更新 email: mary.patterso@classicmodelcars.com

```
UPDATE employees
SET
    email = 'mary.patterson@classicmodelcars.com'
WHERE
    employeeNumber = 1056;
```

2. 修改多列中的值

   同時更新員工編號 1056 的 lastname 和 email 欄位：

```
UPDATE employees
SET
    lastname = 'Hill',
    email = 'mary.hill@classicmodelcars.com'
WHERE
    employeeNumber = 1056;
```

> 客戶 Fresnière Jean 搬到美國了，將他的電話改為 0938-123456

```
UPDATE customers
SET
    phone = '0981-123456',
    country = 'USA'
WHERE
    contactLastName = 'Fresnière'
    AND contactFirstName = 'Jean ';
```

3. 使用 MySQL UPDATE 取代字串

```
SELECT *
FROM employees
WHERE
   jobTitle = 'Sales Rep' AND
   officeCode = 6;
```

更新所有 Sales Reps 具有 officeCode 是 6 的 email domain：

```
UPDATE employees
SET email =
REPLACE(email,'@classicmodelcars.com','@mysqltutorial.org')
WHERE
   jobTitle = 'Sales Rep' AND
   officeCode = 6;
```

> 練習：將 employees 表忠所有 officeCode 是 4 且 reportsTo 為 1102 的員工 email domain 改為 hello.com

```
UPDATE employees
SET email =
REPLACE(email,'@classicmodelcars.com','@hello.com')
WHERE
   reportsTo = '1102' AND
   officeCode = 4;
```

4.使用 UPDATE 更新 SELECT 語句傳回的行範例
SET 可以從 SELECT 查詢其他資料表的資料中提供子句的值。

查詢 customers 表中哪些客戶沒有銷售代表（saleRepEmployeeNumber）。

```
SELECT
    customername,
    salesRepEmployeeNumber
FROM
    customers
WHERE
    salesRepEmployeeNumber IS NULL;
```

從 employees 表中**隨機選擇**一個員工，將其更新到 employees 表中。

首先，測試查詢取得正確員工編號

```
SELECT
    employeeNumber
FROM
    employees
WHERE
    jobtitle = 'Sales Rep'
ORDER BY RAND()
LIMIT 1;
```

將上面的查詢放在 UPDATE 語句 SET 中

```
UPDATE customers
SET
    salesRepEmployeeNumber = (
        SELECT
            employeeNumber
        FROM
            employees
        WHERE
            jobtitle = 'Sales Rep'
        ORDER BY RAND()
        LIMIT 1)
WHERE
    salesRepEmployeeNumber IS NULL;
```

> 練習： 將所有 customers salesRepEmployeeNumber 是員工編號 1504 的員工改為 Bott Larry。

```
UPDATE customers
SET salesRepEmployeeNumber = (
        select employeeNumber
        FROM employees
        WHERE lastName = 'Bott' AND firstName ='Larry'
    )
WHERE salesRepEmployeeNumber = '1504';
```

MYSQL CONSTRAINTS

主鍵 PRIMARY KEY

主鍵是唯一識別表中每一行的一列或一組欄位。

語法：

```
CREATE TABLE table_name(
   column1 datatype PRIMARY KEY,
   column2 datatype,
   ...
);
```

也可以將 `PRIMARY KEY` 放在最後面：

```
CREATE TABLE table_name(
   column1 datatype,
   column2 datatype,
   ...,
   PRIMARY KEY(column1)
);
```

定義多列主鍵

```
CREATE TABLE table_name(
   column1 datatype,
   column2 datatype,
   column3 datatype,
   ...,
   PRIMARY KEY(column1, column2)
);
```

在現有表單中新增主鍵

如果現有表單沒有設定主鍵，可以使用 `ALTER TABLE ... ADD PRIMARY KEY` 來新增主鍵：

```
ALTER TABLE table_name
ADD PRIMARY KEY(column1, column2, ...);
```

刪除主鍵

實際上，很少會刪除主鍵。

```
ALTER TABLE table_name
DROP PRIMARY KEY;
```

範例與練習：

1. 定義單主鍵範例

```
CREATE TABLE products(
   id INT PRIMARY KEY,
   name VARCHAR(255) NOT NULL
);
```

新增幾筆資料：

```
INSERT INTO products (id, name)
VALUES
    (1, 'Laptop'),
    (2, 'Smartphone'),
    (3, 'Wireless Headphones');
```

在主鍵中插入重複值，會出現錯誤訊息。

```
INSERT INTO products (id, name)
VALUES
  (1, 'Bluetooth Speaker');
```

錯誤訊息：

```
ERROR 1062 (23000): Duplicate entry '1' for key 'products.PRIMARY'
```

2. 具有 `AUTO_INCREMENT` 的主鍵範例

```
DROP TABLE products;

CREATE TABLE products(
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255) NOT NULL
);
```

新增幾筆資料：

```
INSERT INTO products (name)
VALUES
    ('Laptop'),
    ('Smartphone'),
    ('Wireless Headphones');
```

MySQL 會自動產生連續的整數值。

3. 定義組合主鍵範例
   建立一個新表單

```
CREATE TABLE customers(
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL
);
```

如果想知道每個客戶喜歡且經常購買的產品有哪些。
為了建立這種關係模型，需要建立一個 favorites 表：

```
CREATE TABLE faviorites(
    customer_id INT,
    product_id INT,
    favorite_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

> 練習：將 customer_id, product_id 設定為組合主鍵。

```
ALTER TABLE faviorites
ADD PRIMARY KEY(customer_id, product_id);
```

4. 新增主鍵

   建立一個 tags 表：

```
CREATE TABLE tags(
    id INT,
    name VARCHAR(25) NOT NULL
);
```

> 練習：將 id 設定成主鍵

```
ALTER TABLE tags
ADD PRIMARY KEY(id);
```

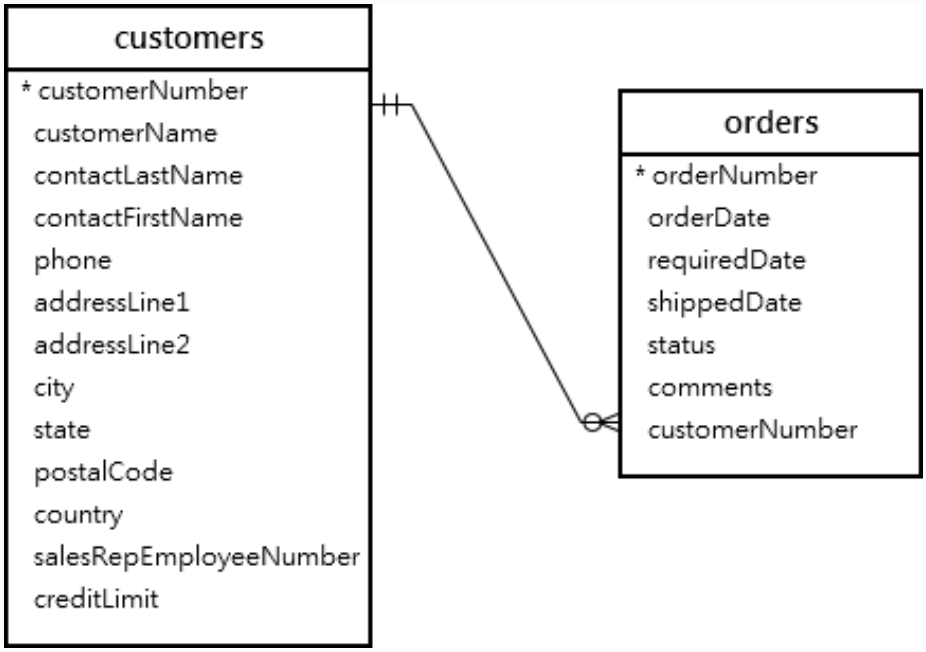
5. 刪除主鍵

> 練習： 刪除 faviorites 表的主鍵

```
ALTER TABLE faviorites
DROP PRIMARY KEY;
```

外鍵 FOREIGN KEY

外鍵是表中的一列或一組列，用來連接到另一個表中的一列或一組列。



圖中，每個客戶可以有零個或多個訂單，並且每個訂單屬於一個客戶。

通常 `Customers` 表稱為 `父表` 或 `被引用表` ， `orders` 表稱為 `子表` 或 `引用表` 。

`子表` 的 `外鍵` 通常是引用 `父表` 的 `主鍵` 。

一個表可以有多個外鍵，每個外鍵引用不同父表的主鍵。

`子表` 中的外鍵必須在父表的主鍵具有對應的行，或者這些外鍵的值必須是NULL。

自引用外鍵

子表和父表可能會引用同一個表。在這種情況下，外鍵會引用同一個表中的主鍵。

`reportTo` 是外鍵，它引用 `employees` 表的 `employeeNumber` 主鍵。

每個員工向**零個或一個**下屬報告，並且一個員工可以有**零個或多個**下屬。

語法：

```
[CONSTRAINT constraint_name]
FOREIGN KEY [foreign_key_name] (column_name, ...)
REFERENCES parent_table(colunm_name,...)
[ON DELETE reference_option]
[ON UPDATE reference_option]
```

- 外鍵約束的名稱 `CONSTRAINT` 。如果省略約束名稱，MySQL 會自動為外鍵約束產生一個名稱。
- `foreign_key_name` 外鍵名稱是可選的，如果省略，則會自動產生。後面指定外鍵欄位以逗號分隔。
- 指定父表，後面跟著外鍵引用，以逗號分隔每個對應欄位。
- `ON DELETE` 指定外鍵如何維護子表和父表之間的參考完整性。
- `ON UPDATE` 決定當父鍵的值被刪除或更新時 MySQL 將採取的動作。

  MySQL 有五個引用選項：

  - CASCADE：如果父表中的某一行被刪除或更新，則子表中符合行的值也會自動刪除或更新。

    > 應用案例：電子商務平台的訂單與訂單明細
    >
    > 當客戶下單後，訂單 (Orders) 表會記錄主要的訂單資訊，而訂單明細 (OrderDetails) 表記錄這筆訂單中包含了哪些商品。

> 如果客戶取消了整個訂單，只需要刪除 Orders 表中的訂單記錄，OrderDetails 表中所有屬於該訂單的明細行也會被自動刪除。這確保了訂單明細不會成為孤立資料，減少了手動維護的麻煩。

- SET NULL：如果父表中的一行被刪除或更新，則子表中外鍵列（或多列）的值將設為NULL。

  > 應用案例：公司部門與員工

> 在一個公司資料庫中，Departments 表記錄所有部門，而 Employees 表則記錄所有員工，每個員工都與一個部門關聯。
>
> 當某個部門被解散或合併。我們會刪除 Departments 表中這個部門，Employees 表中原來在這個部門員工的 department_id 外鍵會自動設定為 NULL。表示這些員工目前沒有部門，但員工的記錄仍然存在，不會被刪除。

- RESTRICT：如果父表中的某行在子表中有符合的行，則 MySQL 會拒絕刪除或更新父表中的行。

> 應用案例：圖書館的書籍與借閱紀錄

> Books 表記錄圖書館所有書籍，Loans 表記錄書籍的借閱情況。

> 一本書籍有多筆借閱紀錄。你嘗試刪除 Books 表中這本書的記錄，因為 Loans 表中仍有這本書的借閱紀錄，RESTRICT 會拒絕這個刪除操作並拋出錯誤。這迫使你必須先處理完所有借閱紀錄（例如，確認所有書籍都已歸還），才能刪除這本書籍的記錄。這可以防止你意外地刪除正在被使用的重要資料。

- NO ACTION：與 RESTRICT 相同。

> MySQL 選擇同時支援這兩個關鍵字，是為了更廣泛地相容於 SQL 標準，讓開發者在從其他資料庫系統（如 PostgreSQL）遷移到 MySQL 時，可以更平滑地轉換，避免語法上的問題。

- SET DEFAULT: 可以被 MySQL 解析器辨識。但是，InnoDB 和 NDB 表都會拒絕此操作。

MySQL **完全支援三種操作**： `RESTRICT` 、 `CASCADE` 和 `SET NULL` 。

如果沒指定 `ON DELETE` 和 `ON UPDATE` ，預設為 `RESTRICT` 。

範例與練習：
建立一個 `fkdemo` 新資料庫。

```
CREATE DATABASE fkdemo;

USE fkdemo;
```

在 `fkdemo` 建立兩個表格 `categories` 和 `products` ：

```
CREATE TABLE categories(
  categoryId INT AUTO_INCREMENT PRIMARY KEY,
  categoryName VARCHAR(100) NOT NULL
) ENGINE = INNODB;
```

```
CREATE TABLE products(
  productId INT AUTO_INCREMENT PRIMARY KEY,
  productName VARCHAR(100) NOT NULL,
  categoryId INT
) ENGINE = INNODB;
```

> 練習：在表 products 增加約束名稱 fk_category，外鍵為 categoryId 並參照表 categories

```
CREATE TABLE products(
  productId INT AUTO_INCREMENT PRIMARY KEY,
  productName VARCHAR(100) NOT NULL,
  categoryId INT,
  CONSTRAINT fk_category FOREIGN KEY (categoryId)
                         REFERENCES categories(categoryId)
) ENGINE = INNODB;
```

因為設定引用，故引用為預設值 `RESTRICT`

1.RESTRICT & NO ACTION actions

1. 在表 categories 新增兩筆資料 Smartphone, Smartwatch

> 練習：

```
INSERT INTO categories(categoryName)
VALUES
('Smartphone'),
('Smartwatch');
```

2. 在表 products 新增一筆資料 iphone，categoryId 為 1
   練習：

```
INSERT INTO products(productName, categoryId)
VALUES('iPhone',1);
```

3. 嘗試插入新資料行

```
INSERT INTO products(productName, categoryId)
VALUES('iPad',3);
```

出現錯誤訊息：

`Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (fkdemo.products, CONSTRAINT fk_category FOREIGN KEY (categoryId) REFERENCES categories(categoryId) ON DELETE RESTRICT ON UPDATE RESTRICT)`

因 `categoryId` 沒有 3

4. 將 `categories` 表欄位 `categoryId` 的值更新為 100：

```
UPDATE categories
SET categoryId = 100
WHERE categoryId = 1;
```

出現錯誤訊息：

`Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails (fkdemo.products, CONSTRAINT fk_category FOREIGN KEY (categoryId) REFERENCES categories (categoryId) ON DELETE RESTRICT ON UPDATE RESTRICT)`

無法刪除或更新，因為 categories 表的 categoryId 已經被 products 表引用。

2. CASCADE
3. 刪除表 products
4. 使用外鍵和選項建立 products 表：ON UPDATE CASCADEON DELETE CASCADE

> 練習：
>
> ```
> CREATE TABLE products(
> productId INT AUTO_INCREMENT PRIMARY KEY,
> productName varchar(100) not null,
> categoryId INT NOT NULL,
> CONSTRAINT fk_category
> FOREIGN KEY (categoryId)
> REFERENCES categories(categoryId)
> ) ENGINE=INNODB;
> ```

```
CREATE TABLE products(
  productId INT AUTO_INCREMENT PRIMARY KEY,
  productName varchar(100) not null,
  categoryId INT NOT NULL,
  CONSTRAINT fk_category
  FOREIGN KEY (categoryId)
  REFERENCES categories(categoryId)
      ON UPDATE CASCADE
      ON DELETE CASCADE
) ENGINE=INNODB;
```

3. 新增資料

```
INSERT INTO products(productName, categoryId)
VALUES
    ('iPhone', 1),
    ('Galaxy Note',1),
    ('Apple Watch',2),
    ('Samsung Galary Watch',2);
```

4. 更新資料

練習：將 categories 表中的 categoryId 1 更新為 100 ：

```
UPDATE categories
SET categoryId = 100
WHERE categoryId = 1;
```

5. 確認更新狀態 `select * from categories`
6. 查詢 products 表 `select * from products`
7. 刪除資料
   練習：從 categories 表中刪除 categoryId 2 ：

```
DELETE FROM categories
WHERE categoryId = 2;
```

8. 確認結果

3. SET NULL
4. 刪除 categories 和 products 表
5. 重新建立 categories 和 products 表

```
CREATE TABLE categories(
    categoryId INT AUTO_INCREMENT PRIMARY KEY,
    categoryName VARCHAR(100) NOT NULL
)ENGINE=INNODB;
```

```
CREATE TABLE products(
    productId INT AUTO_INCREMENT PRIMARY KEY,
    productName varchar(100) not null,
    categoryId INT,
    CONSTRAINT fk_category
    FOREIGN KEY (categoryId)
        REFERENCES categories(categoryId)
        ON UPDATE SET NULL
        ON DELETE SET NULL
)ENGINE=INNODB;
```

3. 在 categories 表中新增資料：

```
INSERT INTO categories(categoryName)
VALUES
    ('Smartphone'),
    ('Smartwatch');
```

4. 在 products 表中新增資料：

```
INSERT INTO products(productName, categoryId)
VALUES
    ('iPhone', 1),
    ('Galaxy Note',1),
    ('Apple Watch',2),
    ('Samsung Galary Watch',2);
```

5. 將 categories 表中的 categoryId 1 更新為 100 ：

```
UPDATE categories
SET categoryId = 100
WHERE categoryId = 1;
```

6. 驗證更新 `SELECT * FROM categories;`

7. 查詢 products 表 `SELECT * FROM products;`

8. 從 categories 表中刪除 categoryId 2：

```
DELETE FROM categories
WHERE categoryId = 2;
```

9. 查詢 products 表 `SELECT * FROM products;`

刪除外鍵

語法：

```
ALTER TABLE table_name
DROP FOREIGN KEY constraint_name;
```

`constraint_name` 是在建立或在表中新增外鍵約束時指定的外鍵約束的名稱。

> 練習：刪除 products 外鍵約束

```
ALTER TABLE products
DROP FOREIGN KEY fk_category;
```

停用外鍵檢查

在停用外鍵約束檢查的情況下，您可以按任意順序將資料載入到父表和子表中。

如果不停用外鍵檢查，則必須先將資料載入到父表中，然後再按順序載入到子表中，這可能會很繁瑣。

另一種需要停用外鍵檢查的情況是刪除表時。除非停用外鍵檢查，否則無法刪除被外鍵約束引用的表。

若要停用外鍵檢查，請將foreign_key_checks變數設為零，如下所示：

```
SET foreign_key_checks = 0;
```

若要啟用外鍵約束檢查，請將值設為foreign_key_checks1：

```
SET foreign_key_checks = 1;
```

範例：

```
CREATE TABLE countries(
    country_id INT AUTO_INCREMENT,
    country_name VARCHAR(255) NOT NULL,
    PRIMARY KEY(country_id)
);

CREATE TABLE cities(
    city_id INT AUTO_INCREMENT,
    city_name VARCHAR(255) NOT NULL,
    country_id INT NOT NULL,
    PRIMARY KEY(city_id),
    FOREIGN KEY(country_id)
    REFERENCES countries(country_id)
);
```

1. 插入一新資料到 cities：

```
INSERT INTO cities(city_name, country_id)
VALUES('New York',1);
```

2. 停用外鍵檢查：

```
SET foreign_key_checks = 0;
```

3. 插入一新資料到 cities
4. 重新啟用外鍵約束檢查

```
SET foreign_key_checks = 1;
```

重新啟用外鍵檢查後，MySQL 不會重新驗證表中的資料。但是，它不允許您插入或更新違反外鍵約束的資料。

5. 在 countries 表中插入一行，其值為 country_id 1，使兩個表中的資料保持一致：

```
INSERT INTO countries(country_id, country_name)
VALUES(1,'USA');
```

刪除具有外鍵約束的資料表

```
DROP TABLE countries;
```

MySQL 發出錯誤

```
(conn:4, no: 1451, SQLState: 23000) Cannot delete or update
a parent row: a foreign key constraint fails sql: DROP
TABLE countries - parameters:[]
```

要解決此問題，有兩個選擇：

- 先移除表 cities，然後移除表 countries。
- 停用外鍵檢查並按任意順序刪除表。

1. 刪除表之前停用外鍵約束檢查

```
SET foreign_key_checks = 0;
```

2. 刪除兩個表 countries 和 cities

```
DROP TABLE countries;
DROP TABLE cities;
```

3. 最後啟用外鍵檢查

```
SET foreign_key_checks = 1;
```

UNIQUE 約束

有時，為了確保一列或一組列中的值是唯一的。例如，使用者的電子郵件地址或客戶的電話號碼應該是唯一的。為了強制執行此規則，可以使用 `UNIQUE` 約束。

語法：

```
CREATE TABLE table_name(
    ...,
    column1 datatype UNIQUE,
    ...
);
```

兩列或多列定義約束語法

```
CREATE TABLE table_name(
   ...
   column1 datatype,
   column2 datatype,
   ...,
   UNIQUE(column1, column2)
);
```

如果定義 `UNIQUE` 約束時如未指定名稱，MySQL 會自動為其產生一個名稱。如果要定義 `UNIQUE` 帶有名稱的約束，使用下列語法：

```
[CONSTRAINT constraint_name]
UNIQUE(column_list)
```

範例：

```
CREATE TABLE suppliers (
    supplier_id INT AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    phone VARCHAR(15) NOT NULL UNIQUE,
    address VARCHAR(255) NOT NULL,
    PRIMARY KEY (supplier_id),
    CONSTRAINT uc_name_address UNIQUE (name,address)
);
```

- `UNIQUE` 在此範例中，為列定義了第一個約束 phone
- 第二個 `UNIQUE` 約束同時包含 `name` 和 `address`

1. 新增一筆資料到 `suppliers` 。

```
INSERT INTO suppliers(name, phone, address)
VALUES( 'ABC Inc',
       '(408)-908-2476',
       '4000 North 1st Street');
```

2. 新增一筆不同的供應商，但相同的電話號碼到 `suppliers` 表中。

```
INSERT INTO suppliers(name, phone, address)
VALUES( 'XYZ Corporation','(408)-908-2476','3000 North 1st
Street');
```

MySQL 發出錯誤：

```
Error: (conn:17, no: 1062, SQLState: 23000) Duplicate entry
'(408)-908-2476' for key 'phone' sql: INSERT INTO
suppliers(name, phone, address) VALUES( 'XYZ
Corporation','(408)-908-2476','3000 North 1st Street') -
parameters:[]
```

3. 換個電話號碼

```
INSERT INTO suppliers(name, phone, address)
VALUES( 'XYZ Corporation','(408)-908-3333','3000 North 1st
Street');
```

4. 在 `suppliers` 表中插入一行已存在於列的 `name` 和 `address`

```
INSERT INTO suppliers(name, phone, address)
VALUES( 'ABC Inc',
       '(408)-908-1111',
       '4000 North 1st Street');
```

MySQL 發出錯誤：

```
Error: (conn:17, no: 1062, SQLState: 23000) Duplicate entry
'ABC Inc-4000 North 1st Street' for key 'uc_name_address'
sql: INSERT INTO suppliers(name, phone, address) VALUES(
'ABC Inc', '(408)-908-1111', '4000 North 1st Street') -
parameters:[]
```

> **constraint_name 作用**
> 命名在資料庫的管理和維護提供了很大的便利性，主要體現在以下幾個方面：
>
> 1. 易於管理和維護
>
> 當需要對資料庫進行修改或維護時，有意義的約束名稱可以快速識別和定位。
>
> - **範例：**
>   - 如果沒有命名，MySQL 會自動生成一個類似 `idx_1a2b3c4d5e` 的隨機名稱。這個名稱既沒有意義也難以記憶。
>   - 如果命名為 `uq_products_sku`，一眼就能看出這個約束是為了確保 `products` 表格中的 `sku`（商品庫存單位）欄位是唯一的。
>
> 2. 方便刪除和修改
>
> 當需要**刪除**這個唯一性約束時，必須知道它的名稱。如果沒有命名，得先查詢資料庫的元資料（metadata）才能找到它自動生成的名稱。

- **範例：**
  - 有命名：`ALTER TABLE products DROP CONSTRAINT uq_products_sku;`
  - 沒有命名：必須先執行 `SHOW CREATE TABLE products;` 找出自動生成的名稱，然後再執行 `ALTER TABLE products DROP CONSTRAINT idx_1a2b3c4d5e;`。

3. 清晰的錯誤訊息

當資料庫操作因為違反唯一性約束而失敗時，錯誤訊息中會包含約束的名稱。一個清晰的名稱能幫助我們或開發人員**立即理解錯誤原因**，而不需要再去查閱資料庫結構。

- **範例：**
  - 訊息：`Duplicate entry 'sku123' for key 'uq_products_sku'`
  - 讀者馬上就能知道：是因為嘗試插入重複的 `sku123`，且這個操作被名為 `uq_products_sku` 的約束阻止了。

UNIQUE 與 NULL

在 MySQL 中，當涉及 `UNIQUE` 時，NULL 值被視為不同的值。

建立一個 `contacts` 新表

```
CREATE TABLE contacts(
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    phone VARCHAR(20) UNIQUE
)
```

在 `contacts` 表中插入一些資料

```
INSERT INTO contacts(name, phone)
VALUES
    ('Alice','(408)-102-2456'),
    ('John', NULL),
    ('Jane', NULL);
```

在電話列中插入兩個 NULL 值不會導致重複。

檢索 `contacts` 資料

```
SELECT * FROM contacts;
```

UNIQUE 與 INDEX

當一列或一組列被定義唯一約束時，MySQL 會建立對應的 UNIQUE INDEX 並使用該 `INDEX` 來強制執行規則。

1. 使用 `SHOW CREATE TABLE` 顯示 `suppliers` 表格的定義：

語法：

```
SHOW CREATE TABLE suppliers;
```

`UNIQUE` 在 `suppliers` 表上創建了兩個索引：phone 和 uc_name_address。

2. 使用 `SHOW INDEX` 顯示與 `suppliers` 資料表相關的所有索引。

語法：

```
SHOW INDEX FROM suppliers;
```

刪除 UNIQUE 約束

要刪除 `UNIQUE` 約束，可以使用 `DROP INDEX` 或 `ALTER TABLE`。

```
DROP INDEX index_name ON table_name;
```

或

```
ALTER TABLE table_name
DROP INDEX index_name;
```

> 練習：刪除 `suppliers` 表上的 uc_name_address 約束

```
DROP INDEX uc_name_address ON suppliers;
```

新增的 UNIQUE 約束

使用 ALTER TABLE ADD CONSTRAINT 在現有表的列新增 UNIQUE 約束

語法：

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
UNIQUE (column_list);
```

> 練習：重新將 uc_name_address 約束加回 suppliers 表

```
ALTER TABLE suppliers
ADD CONSTRAINT uc_name_address
UNIQUE (name, address);
```

NOT NULL 約束

NOT NULL確保儲存在列中的值不為NULL。

語法：

```
column_name data_type NOT NULL;
```

`NOT NULL` 約束強制執行該列不能包含任何 `NULL` 值。

如果嘗試**更新**或**插入** `NULL` 值到 `NOT NULL` 欄位中，MySQL 將發出錯誤。

使用 CREATE TABLE 建立 tasks 表格：

```
CREATE TABLE tasks (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE
);
```

在表中，我們明確定義了帶有 `NOT NULL` 約束的 `title` 和 `start_date` 列。

而 `id` 列具有 `PRIMARY KEY` 約束，因此，它也包含一個 `NOT NULL` 約束。

`end_date` 列可以是 NULL ，因為在建立新任務時，可能不知道完成日期

檢視結構

```
DESC tasks;
```

除非有特殊原因，否則最好每一列都設定 `NOT NULL` 約束。因為，NULL值會讓查詢複雜化。

在現有欄位中新增 NOT NULL 約束

新增一些資料到 tasks

```
INSERT INTO tasks(title ,start_date, end_date)
VALUES('Learn MySQL NOT NULL constraint', '2017-02-
01','2017-02-02'),
      ('Check and update NOT NULL constraint to your
database', '2017-02-01',NULL);
```

1. 檢查該欄位的目前值是否有任何 NULL。
   使用 `IS NULL` 函式查找 `end_date` 列中包含 `NULL` 的行

```
SELECT * FROM tasks
WHERE end_date IS NULL;
```

2. 更新 NULL 為非 NULL。

```
UPDATE tasks
SET end_date = start_date + 7
WHERE end_date IS NULL;
```

3. 修改具有 NOT NULL 約束的欄位。
   語法：

```
ALTER TABLE table_name
CHANGE
   old_column_name
   new_column_name column_definition;
```

```
ALTER TABLE tasks
CHANGE
    end_date
    end_date DATE NOT NULL;
```

刪除 NOT NULL 約束

刪除 `NOT NULL` 約束使用 `ALTER TABLE..MODIFY`

```
ALTER TABLE table_name
MODIFY column_name column_definition;
```

> 練習：從 tasks 表中的 end_date 欄位刪除 NOT NULL 約束

```
ALTER TABLE tasks
MODIFY end_date DATE;
```

DEFAULT 約束

DEFAULT 約束允許為某一列指定預設值。

語法:

```
column_name data_type DEFAULT default_value;
```

- default_value 是文字常數，例如數字或字串。不能是函數或表達式。但是，MySQL 允許將目前日期和時間（CURRENT_TIMESTAMP）設定為 TIMESTAMP 和 DATETIME。
- 當定義一個不是 `NOT NULL` 約束的欄位時，該列會自動使用 `NULL` 為預設值。
- 如果某個欄位有 DEFAULT 約束，INSERT 或 UPDATE 沒有為該列指定值，則 MySQL 將使用 DEFAULT 約束中指定的預設值。

建立一個 cart_items 的新表，該表包含四個列：item_id、name、quantity、 和 sales_tax

```
CREATE TABLE cart_items
(
    item_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    quantity INT NOT NULL,
    price DEC(5,2) NOT NULL,
    sales_tax DEC(5,2) NOT NULL DEFAULT 0.1,
    CHECK(quantity > 0),
    CHECK(sales_tax >= 0)
);
```

新增一筆記錄到 cart_items

```
INSERT INTO cart_items(name, quantity, price)
VALUES('Keyboard', 1, 50);
```

新增 DEFAULT 約束

為現有表單的欄位新增預設約束，使用 `ALTER TABLE`

語法：

```
ALTER TABLE table_name
ALTER COLUMN column_name SET DEFAULT default_value;
```

將表格 cart_itesm 的 quantity 列新增 DEFAULT 1 約束

```
ALTER TABLE cart_items
ALTER COLUMN quantity SET DEFAULT 1;
```

刪除 DEFAULT 約束

語法：

```
ALTER TABLE table_name
ALTER column_name DROP DEFAULT;
```

從 cart_items 表格的 quantity 刪除 DEFAULT 約束

```
ALTER TABLE cart_items
ALTER quantity DROP DEFAULT;
```

CHECK 約束

在 MySQL 8.0.16 之前，CREATE TABLE 允許新增表格 CHECK 約束。但 MySQL 會忽略所有 CHECK 約束。

從 MySQL 8.0.16 開始，所有儲存引擎 CREATE TABLE 都支援表格和欄位限制的基本特性。

語法：

```
CONSTRAINT constraint_name
CHECK (expression)
[ENFORCED | NOT ENFORCED]
```

在此語法中：

1. 在關鍵字後面指定要建立的檢查約束的名稱 CONSTRAINT。如果省略約束名稱，MySQL 會自動產生一個遵循下列約定的名稱：

   ```
   table_name_chk_n
   ```

   n 是序數，如 1、2 和 3。例如，parts 表格 CHECK 的約束的自動產生名稱會是 parts_chk_1、parts_chk_2 等等。
2. 指定一個布林值 expression，該布林值必須對關鍵字後括號內的表的每一行進行計算。

   > 注意，MySQL 將 1 視為 TRUE，將 0 視為 FALSE。

3. 可選擇性地指定強制執行子句來指示檢查約束是否被強制執行：
    - 使用 ENFORCED 或省略 ENFORCED 來建立和強制執行約束。
    - 用於 NOT ENFORCED 建立約束但不強制執行。
      如前所述，可以將 CHECK 約束定義為表格約束或列約束。

    表格 CHECK 約束可以引用多個列，而列 CHECK 約束可以引用定義它的唯一列。

範例：

1. 建立 CHECK 約束作為列約束
   建立一個 parts 表：

```
CREATE TABLE parts (
    part_no VARCHAR(18) PRIMARY KEY,
    description VARCHAR(40),
    cost DECIMAL(10,2 ) NOT NULL CHECK (cost >= 0),
    price DECIMAL(10,2) NOT NULL CHECK (price >= 0)
);
```

   零件表有兩個欄位設定 CHECK 約束：一個是 cost，另一個是 price。

   因為我們沒有明確指定 CHECK 約束的名稱，所以 MySQL 會自動為它們產生名稱。

   若要查看具有約束名稱的表定義CHECK，請使用下列SHOW CREATE TABLE語句：

```
SHOW CREATE TABLE parts;
```

   MySQL為檢查約束產生了名稱（ `parts_chk_1` 和 `parts_chk_2` ）。

   在 parts 表新增紀錄：

```
INSERT INTO parts(part_no, description,cost,price)
VALUES('A-001','Cooler',0,-100);
```

   MySQL 發出錯誤

```
Error: (conn:46, no: 4025, SQLState: 23000) CONSTRAINT
parts.price failed for hello.parts sql: INSERT INTO
parts(part_no, description,cost,price) VALUES('A-
001','Cooler',0,-100) - parameters:[]
```

2. 建立 CHECK 約束作為表的約束

   刪除 parts 表：

```
DROP TABLE IF EXISTS parts;
```

   重新建立一個 parts 新表：

```
CREATE TABLE parts (
    part_no VARCHAR(18) PRIMARY KEY,
    description VARCHAR(40),
    cost DECIMAL(10,2 ) NOT NULL CHECK (cost >= 0),
    price DECIMAL(10,2) NOT NULL CHECK (price >= 0),
    CONSTRAINT parts_chk_price_gt_cost
        CHECK(price >= cost)
);
```

   檢視 parts 定義：

```
SHOW CREATE TABLE parts;
```

   建立一筆價格低於成本的新零件

```
INSERT INTO parts(part_no, description,cost,price)
VALUES('A-001','Cooler',200,100);
```

新增 CHECK 約束

語法：

```
ALTER TABLE table_name
ADD CHECK (expression);
```

如果要明確指定 CHECK 約束的名稱，可以使用以下 ALTER TABLE ... ADD CONSTRAINT ... CHECK

```
ALTER TABLE table_name
ADD CONSTRAINT contraint_name
CHECK (expression);
```

為 parts 表格新增 CHECK 約束

```
ALTER TABLE parts
ADD CHECK (part_no <> description);
```

新增一筆紀錄

```
INSERT INTO parts
VALUES('A','A',100,120);
```

刪除檢查約束

從表中刪除約束，使用 `ALTER TABLE ... DROP CHECK`

```
ALTER TABLE table_name
DROP CHECK constraint_name;
```

例如，從 parts 表中刪除 CHECK 約束 parts_chk_3

```
ALTER TABLE parts
DROP CHECK parts_chk_3;
```

查詢目前使用的 MYSQL 資料庫編碼：

1. 查詢當前資料庫的編碼

如果你已經連線到特定的資料庫，可以直接執行以下指令：

```
USE your_database_name;
SELECT @@character_set_database, @@collation_database;
```

- `your_database_name`：替換成你實際的資料庫名稱。
- `character_set_database`：會顯示資料庫的預設**字元集 (Character Set)**。
- `collation_database`：會顯示資料庫的預設**排序規則 (Collation)**。

2. 查詢所有資料庫的編碼

如果你想查看伺服器上所有資料庫的編碼設定，可以使用 `information_schema` 資料庫：

```
SELECT SCHEMA_NAME AS db_name,
       DEFAULT_CHARACTER_SET_NAME AS character_set,
       DEFAULT_COLLATION_NAME AS collation
FROM INFORMATION_SCHEMA.SCHEMATA;
```

3. 查詢連線層級的編碼

MySQL 有許多不同層級的編碼設定，例如伺服器層級、資料庫層級、表格層級，以及客戶端連線層級。如果想知道當前連線的編碼設定，可以執行：

```
SHOW VARIABLES LIKE 'character_set%';
```

這會顯示一系列與連線編碼相關的變數，例如：

- `character_set_client`：客戶端傳送指令的編碼。

- `character_set_connection`：伺服器處理指令時使用的編碼。
- `character_set_results`：伺服器回傳結果給客戶端的編碼。

字元集 (Character Set) 定義了資料庫能儲存哪些字元，而 **排序規則 (Collation)** 則定義了這些字元如何進行比較和排序。一般來說，為了完整支援多國語言和表情符號，建議使用 `utf8mb4` 作為字元集，搭配 `utf8mb4_unicode_ci` 或 `utf8mb4_general_ci` 等排序規則。