資料庫概論

什麼是資料模型

- 資料模型是用來描述、組織和管理資料結構的概念性框架或抽象模式。它
- 定義了資料的格式、類型、關係、限制條件以及操作方式，作為資料庫設計和實作的基礎。
- 簡單來說，資料模型是一套規則和概念，用來表達真實世界的資訊，並將其轉換為可被電腦系統處理的形式。

為什麼需要資料模型

1. **降低複雜度**
    - 真實世界的資料結構通常極為複雜，資料模型提供一種將此複雜性簡化和抽象化的方法
    - **實例**：銀行系統中，客戶、帳戶、交易等複雜關係透過資料模型變得清晰可管理
2. **溝通工具**
    - 作為技術人員和非技術人員之間的溝通橋樑
    - **實例**：系統分析師可以使用ER模型圖與業務部門討論需求，確保雙方對資料需求有共同理解
3. **確保資料一致性與完整性**
    - 定義資料之間的關係和限制條件，防止錯誤或不一致的資料進入系統
    - **實例**：在學生資料庫中，設定學號為主鍵，確保不會有重複的學生記錄
4. **資料共享與整合**
    - 提供標準化的資料表示和組織方式，使不同應用程式能共享和整合資料
    - **實例**：企業中的ERP系統與CRM系統可以共享客戶資料，減少重複輸入和不一致
5. **系統設計與開發指南**
    - 提供資料庫設計和應用程式開發的藍圖
    - **實例**：開發團隊根據關聯式資料模型設計數據表結構，確定各表間的關係和索引策略
6. **系統靈活性與可擴展性**
    - 良好的資料模型設計能支援系統的未來擴展，減少改動成本
    - **實例**：電子商務平台初期只銷售實體商品，資料模型考慮未來可能銷售數位商品，預留相應結構
7. **效能優化**
    - 合適的資料模型能優化資料存取和處理效率
    - **實例**：根據查詢需求設計的星狀模型(Star Schema)能大幅提升資料倉儲的查詢效能
8. **維護和管理成本降低**
    - 清晰的資料模型使系統更易於理解、維護和管理
    - **實例**：當需要修改系統功能時，開發人員能迅速理解現有資料結構，減少錯誤和開發時間

資料模型如同建築藍圖，它不僅描述了資料的組織方式，也決定了整個系統的結構、效能和可擴展性。一個設計良好的資料模型能確保系統長期穩定運行，並能適應不斷變化的業務需求。

資料模型簡介

2.1.1 資料模型的分類

資料模型是描述資料庫結構的方法，主要可分為三大類：

1. **概念資料模型 (Conceptual Data Models)**
    - 提供較高層次、接近人類思考方式的資料觀點
    - 專注於描述資料的結構而非儲存細節
    - **實例**：設計一個大學資料庫系統，ER 模型中會包含「學生」、「課程」和「教授」等實體，以及「選修」和「教授」等關係，每個實體有屬性如學生ID、姓名、課程代碼等
    - 主要用於資料庫設計的初期階段，與使用者溝通需求
2. **邏輯資料模型 (Logical Data Models)**
    - 介於概念模型和實體模型之間的中間層
    - 描述資料將如何在資料庫系統中表示，但不涉及實際儲存細節
    - 主要分類：
        - 關聯式模型 (Relational Model)：實例：將上述大學系統轉換為關聯式模型，會創建表格如「學生表」(student_id, name, major)、「課程表」(course_id, title, credits)和「選修表」(student_id, course_id, semester, grade)
        - 網狀模型 (Network Model)：實例：在訂單管理系統中，客戶記錄直接連結到多個訂單記錄，訂單記錄又連結到多個產品記錄
        - 階層式模型 (Hierarchical Model)：實例：組織結構中，CEO下有多個部門主管，部門主管下有多個團隊領導，呈樹狀結構
        - 物件導向模型 (Object-Oriented Model)：實例：在學生管理系統中，定義學生類別(Student Class)含屬性和方法，每個學生實例都繼承這些特性
        - 文件導向模型 (Document Model)：實例：MongoDB中存儲的商品資料，一個文件包含商品基本資訊、多張圖片URLs和用戶評論等多層次資料
3. **實體資料模型 (Physical Data Models)**
    - 描述資料在電腦儲存媒體上的實際存放方式
    - **實例**：決定學生表使用B+樹索引加速學號查詢，課程資料使用雜湊索引(Hash Index)加速課程代碼查詢，大型文字資料（如課程描述）使用壓縮儲存節省空間
    - 包含檔案組織、索引結構、記憶體分配、壓縮技術等細節
    - 主要關注效能、儲存空間優化和存取效率

2.1.2 資料獨立性 (Data Independence) 的觀念

資料獨立性是資料庫管理系統的核心特性，意指應用程式和上層結構不需因下層資料結構的改變而修改。主要分為兩種層次：

1. **邏輯資料獨立性 (Logical Data Independence)**
    - 保護應用程式不受邏輯資料模型變更的影響
    - **實例**：假設原本學生資料表只有基本資訊，現在需要增加「緊急聯絡人」欄位，透過邏輯資料獨立性，使用原本學生資料的應用程式（如成績計算系統）不需要修改程式碼，仍可正常運作

    - **實例**：圖書館系統中，決定將原本的「書籍」表拆分為「書籍基本資料」和「書籍庫存資訊」兩個表，但查詢介面仍維持原貌，用戶不需學習新的使用方式
    - 實現方式：透過資料庫視圖(Views)提供穩定的資料存取介面，隱藏內部結構變更
2. **實體資料獨立性 (Physical Data Independence)**
    - 保護邏輯結構不受實體儲存結構變更的影響
    - **實例**：資料庫管理員決定為「訂單」表新增訂單日期的索引以加速日期範圍查詢，此變更不影響應用程式的查詢語法及邏輯結構
    - **實例**：原本將影像資料直接存在資料庫表中，後來改為存儲檔案路徑並將實際影像存在專用伺服器，應用程式透過相同介面存取，不需修改
    - **實例**：將熱門存取的資料從硬碟遷移到快速的SSD或記憶體資料庫，提升系統效能，但查詢方式不變
    - 實現方式：透過資料庫管理系統的抽象層和內部映射機制

資料獨立性的具體實例應用：

- **系統升級案例**：APPLE 將資料庫從 MS SQL 遷移到 PostgreSQL，因為良好的邏輯獨立性設計，前端應用系統（網路銀行、ATM系統等）幾乎不需要修改
- **效能最佳化案例**：電子商務平台在節慶前將熱門商品資料從傳統資料庫移至 Redis 快取，系統行為和用戶體驗保持一致，但處理速度大幅提升
- **業務擴展案例**：CRM 系統原本只記錄客戶基本資料，隨業務發展需增加社交媒體互動記錄，透過資料獨立性設計，原有的客戶查詢和報表功能不受影響，同時成功整合新資料

這些實例展示了資料模型和資料獨立性如何在實際系統中應用，提供系統彈性和可維護性。

傳統的資料模型說明與實例

2.2.1 階層式的資料模型 (hierarchical data model)

階層式資料模型的基本觀念

階層式資料模型是最早期的資料庫模型之一，於1960年代開發。這種模型使用樹狀結構來表示資料之間的關係，類似於一個倒置的樹，具有以下特點：

1. **父子關係**：每個記錄（除了根節點）只能有一個父節點，但可以有多個子節點
2. **單一路徑**：從根節點到任何記錄只有一條唯一路徑
3. **一對多關係**：只能表示一對多（1:N）的關係
4. **根節點**：資料結構由單一的根節點開始

實際範例：**IBM的IMS (Information Management System)**

假設一家公司的組織結構：

```
部門
 |
 ├── 員工1
 |      └── 技能1
 |      └── 技能2
 |
 ├── 員工2
 |      └── 技能1
 |
 └── 員工3
        └── 技能3
        └── 技能4
```

在此模型中，一個部門可以有多個員工，一個員工可以有多種技能，但每個員工只能屬於一個部門，每種技能只能綁定於一個員工。

階層式資料模型中資料的處理

階層式資料庫主要使用以下技術處理資料：

1. **導航式存取**：透過樹狀結構進行遍歷
    - 向下移動：從父記錄到子記錄
    - 向上移動：從子記錄到父記錄
    - 水平移動：在同層次的記錄間移動
2. **操作指令**：
    - GET UNIQUE：取得特定記錄
    - GET NEXT：取得下一筆記錄
    - GET NEXT WITHIN PARENT：取得同一父節點下的下一筆記錄
    - INSERT：新增記錄
    - DELETE：刪除記錄
    - REPLACE：更新記錄

實際應用範例：**醫院病患資料管理**

```
醫院
 |
 ├── 部門1 (內科)
 |      ├── 醫生A
 |      |      └── 病患1
 |      |      └── 病患2
 |      |
 |      └── 醫生B
 |             └── 病患3
 |
 └── 部門2 (外科)
        ├── 醫生C
        |      └── 病患4
        |
```

```
    └── 醫生D
        └── 病患5
```

要查詢「醫生A的所有病患」，系統需要：

1. 找到根節點（醫院）
2. 導航到「部門1」
3. 導航到「醫生A」
4. 讀取其所有「病患」子節點

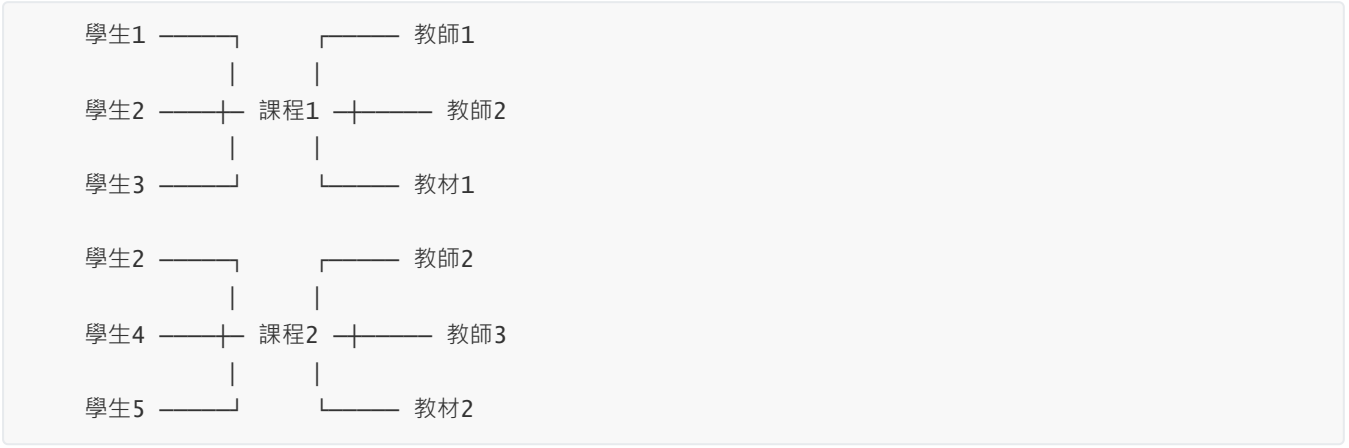
2.2.2 網路式的資料模型 (network data model)

網路資料模型的基本觀念

網路式資料模型是對階層式模型的擴展，由CODASYL（Conference on Data Systems Languages）於1969年提出。其關鍵特點：

1. **多對多關係**：能表示多對多（M:N）的關係
2. **記錄類型**：資料組織為記錄類型（record type）
3. **集合類型**：使用集合類型（set type）表示記錄間的關係
4. **擁有者-成員關係**：每個集合由一個擁有者記錄（owner）和多個成員記錄（member）組成
5. **複雜連結**：一個記錄可以同時是多個集合的成員

實際範例：**大學選課系統**



此範例中，學生和課程之間是多對多關係（一個學生可修多門課，一門課可有多名學生）；教師和課程之間也是多對多關係（一位教師可教多門課，一門課可有多位教師）。

網路資料模型中資料的處理

網路模型的資料處理方式包括：

1. **記錄導航**：
   - FIND：找到特定記錄
   - GET：獲取記錄內容
   - FIND OWNER：找到集合的擁有者
   - FIND NEXT/PRIOR/FIRST/LAST：在集合中導航

2. **典型操作流程**：
    - 先找到特定記錄（通常是擁有者）
    - 然後導航到相關的集合成員
    - 再處理這些成員記錄
3. **資料操作命令**：
    - STORE：新增記錄
    - ERASE：刪除記錄
    - MODIFY：修改記錄
    - CONNECT：將記錄連接到集合
    - DISCONNECT：從集合中斷開記錄

實際應用範例：**訂單管理系統**

```
          ┌── 產品1
          |
客戶1 ──┼── 產品2
          |
          └── 產品3

客戶1 ──┬── 訂單1 ──┬── 產品1
          |              └── 產品2
          |
          └── 訂單2 ──┬── 產品2
                         └── 產品3

客戶2 ──── 訂單3 ──┬── 產品1
                         └── 產品3
```

在這個網路模型中：

- 客戶和產品之間可以有直接關係（客戶喜好的產品）
- 客戶和訂單之間是一對多關係
- 訂單和產品之間是多對多關係
- 一個產品可以出現在多個訂單中
- 一個訂單可以包含多個產品

要處理「查找客戶1的所有訂單中包含產品2的訂單」，系統需要：

1. FIND 客戶1記錄
2. FIND FIRST 訂單（在客戶-訂單集合中）
3. 循環處理每個訂單：
    - FIND 產品2（在訂單-產品集合中）
    - 如果找到，記錄此訂單
    - FIND NEXT 訂單（繼續遍歷）

2.3 進階資料模型 (network data model)

2.3.1 實體體關係模型

ER Model 中的實體類型差異

Entity（實體）的基本定義

實體是 ER 模型中用來代表現實世界中獨立存在的物件、概念或事物。每個實體都有屬性（attributes）來描述其特徵。

Strong Entity（強實體）

定義

- 強實體是能獨立存在的實體，不依賴於其他實體的存在
- 擁有自己的主鍵（Primary Key）可唯一識別每個實例
- 在 ER 圖中通常用單線矩形表示

例子

1. **學生（Student）**：以學號作為主鍵，不依賴其他實體存在
2. **員工（Employee）**：以員工編號作為主鍵，獨立存在
3. **產品（Product）**：以產品編號作為主鍵，不依賴其他實體
4. **公司（Company）**：以統一編號作為主鍵，獨立存在
5. **客戶（Customer）**：以客戶編號作為主鍵，不依賴其他實體

Weak Entity（弱實體）

定義

- 弱實體無法獨立存在，依賴於另一個強實體（稱為擁有實體或識別實體）
- 沒有足夠的屬性形成主鍵，需要借用關聯實體的主鍵來共同形成識別
- 在 ER 圖中通常用雙線矩形表示，與擁有實體的關係用雙線菱形表示

例子

1. **銀行帳戶交易（Account Transaction）**：依賴於帳戶（Account）實體存在，交易編號只有在特定帳戶下才有意義
2. **員工眷屬（Employee Dependent）**：依賴於員工實體，眷屬資訊必須關聯到特定員工才有意義
3. **訂單明細項目（Order Line Item）**：依賴於訂單（Order）實體，明細項目必須屬於特定訂單
4. **考試答案（Exam Answer）**：依賴於考試（Exam）實體，答案必須關聯到特定考試
5. **房間（Room）**：依賴於建築物（Building）實體，房號只有在特定建築物才是唯一的

主要差異比較

特性	Strong Entity（強實體）	Weak Entity（弱實體）
存在依賴	獨立存在	依賴於擁有實體

特性	Strong Entity（強實體）	Weak Entity（弱實體）
主鍵	有自己的主鍵	部分鍵（需結合擁有實體的主鍵）
識別方式	透過自身屬性唯一識別	需結合擁有實體的主鍵共同識別
表示方法	單線矩形	雙線矩形
關係類型	各種關係	通常是識別關係（雙線菱形）
刪除規則	可獨立刪除（視關係而定）	當擁有實體刪除時必須一併刪除

適當地識別實體類型對建立正確的資料庫結構、維護資料完整性以及反映業務邏輯非常重要。

Null 與 Empty 在 ER 模型中的差別

在資料庫和 ER (實體關係) 模型中，null 和 empty 代表不同的概念，這些差異很重要。以下是說明這兩個概念差異的實際範例：

1. 聯絡資訊範例

情境：在顧客資料表中記錄電話號碼和電子郵件

- **Null**：顧客 A 的電話號碼欄位為 null，表示「未知」或「不適用」—我們不知道顧客有沒有電話號碼
- **Empty**：顧客 B 的電話號碼欄位為 ""（空字串），表示我們知道顧客確實沒有提供電話號碼

2. 學生成績範例

情境：在學生成績資料表中記錄考試分數

- **Null**：學生未參加考試，所以分數欄位為 null—分數不存在
- **Empty**：學生參加考試但得了0分，所以分數欄位是 0—分數存在但為零

3. 產品描述範例

情境：在產品目錄中記錄產品描述

- **Null**：新產品尚未編寫描述，描述欄位為 null—資訊尚未提供
- **Empty**：產品確實沒有額外描述需求，描述欄位為 ""—有意留空

4. 訂單跟蹤範例

情境：在訂單系統記錄送貨日期

- **Null**：訂單尚未出貨，送貨日期欄位為 null—事件尚未發生
- **Empty**：不適用於日期欄位，日期通常不會是空值而是 null 或具體日期

5. 部門主管範例

情境：在員工資料表中記錄部門主管

- **Null**：該部門還沒指派主管，主管 ID 欄位為 null—資訊不存在
- **Empty**：不適用於 ID 欄位，ID 通常不會是空值而是 null 或具體值

在 SQL 中的查詢差異

```sql
-- 查找 null 值
SELECT * FROM customers WHERE phone_number IS NULL;

-- 查找空值
SELECT * FROM customers WHERE phone_number = '';
```

這些範例說明了 null（未知、不存在或不適用的值）和 empty（存在但內容為空的值）在 ER 模型和資料庫中的根本差異。正確區分這兩種概念對資料建模和查詢都非常重要。

關聯式資料庫

Super Key 超鍵的詳細說明與實例

Super Key 定義

Super Key（超鍵）是關聯式資料庫中的一個重要概念，它指的是**能夠唯一識別資料表中每一筆記錄的屬性或屬性組合**。Super Key 具有唯一性，但不一定是最小的屬性集合。

Super Key 的特性

1. **唯一識別性**：能夠唯一標識資料表中的每一筆記錄
2. **包含性**：可能包含多餘的屬性（非極小）
3. **階層性**：一個資料表可以有多個 Super Key

與其他鍵的關係

- **候選鍵（Candidate Key）**：是不包含多餘屬性的最小 Super Key
- **主鍵（Primary Key）**：從候選鍵中選擇的一個作為主要識別屬性
- **外鍵（Foreign Key）**：參照其他資料表主鍵的屬性

實際實例

實例一：學生資料表

假設有學生資料表 `Students` 包含以下屬性：

- `student_id` (學號)
- `national_id` (身分證號碼)
- `email` (電子郵件)
- `name` (姓名)
- `department` (系所)

Super Keys 可能有：

- {`student_id`}
- {`national_id`}
- {`email`}
- {`student_id`, `name`}
- {`national_id`, `department`}
- {`student_id`, `national_id`, `email`}
- ...以及任何包含上述組合的屬性集合

在這個例子中，`student_id`、`national_id` 和 `email` 各自都可以形成 Super Key，因為它們都能唯一識別一名學生。而 {`student_id`, `name`} 雖然也是 Super Key，但它包含了非必要的 `name` 屬性。

實例二：訂單明細資料表

假設有訂單明細資料表 `OrderDetails` 包含以下屬性：

- `order_id` (訂單編號)
- `product_id` (產品編號)
- `quantity` (數量)
- `unit_price` (單價)
- `discount` (折扣)

Super Keys 可能有：

- {`order_id`, `product_id`}
- {`order_id`, `product_id`, `quantity`}
- {`order_id`, `product_id`, `unit_price`}
- {`order_id`, `product_id`, `discount`}
- {`order_id`, `product_id`, `quantity`, `unit_price`}
- ...以及任何包含 {`order_id`, `product_id`} 的屬性組合

在這個例子中，單一個 `order_id` 或單一個 `product_id` 都不足以唯一識別一筆訂單明細記錄，因為一個訂單可能包含多個產品，一個產品也可能出現在多個訂單中。只有 {`order_id`, `product_id`} 的組合才能唯一識別一筆訂單明細記錄。

實例三：航班資料表

假設有航班資料表 `Flights` 包含以下屬性：

- `flight_number` (航班號碼)
- `departure_date` (出發日期)
- `origin` (出發地)
- `destination` (目的地)
- `aircraft_id` (飛機編號)

Super Keys 可能有：

- {`flight_number`, `departure_date`}
- {`flight_number`, `departure_date`, `origin`}
- {`flight_number`, `departure_date`, `destination`}
- {`flight_number`, `departure_date`, `origin`, `destination`}
- ...以及任何包含 {`flight_number`, `departure_date`} 的屬性組合

在這個例子中，同一個航班號碼（如 CA123）可能每天都有，因此需要結合出發日期才能唯一識別一個特定航班。

Super Key 在資料庫設計中的重要性

1. **資料完整性**：確保資料的唯一性和一致性
2. **查詢效率**：基於 Super Key 建立索引可以提高查詢效能
3. **關聯基礎**：為資料表間的關聯提供基礎
4. **候選鍵選擇**：Super Key 是識別候選鍵的前提，進而選擇主鍵

理解 Super Key 概念有助於正確設計資料庫結構、確保實體完整性，以及優化資料庫性能。

以下是關於 **主鍵（Primary Key）**、**外鍵（Foreign Key）**、**複合鍵（Composite Key）** 和 **候選鍵（Candidate Key）** 的詳細說明與範例：

---

1. 主鍵（Primary Key, PK）

定義

主鍵是資料表中唯一標識每一筆資料的欄位或欄位組合，具有以下特性：

- **唯一性**：每筆資料的主鍵值不可重複。
- **不可為空值（NULL）**。
- **一個資料表只能有一個主鍵**。

範例

在「學生資料表」中，`學號 (StudentID)` 是主鍵，確保每名學生有唯一識別碼：

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(50),
    Email VARCHAR(100)
);
```

資料範例：

StudentID	Name	Email
S001	Alice	alice@email.com
S002	Bob	bob@email.com

---

2. 外鍵（Foreign Key, FK）

定義

外鍵是資料表中指向另一資料表主鍵的欄位，用於建立表之間的關聯，確保 **參照完整性（Referential Integrity）**。

- **值必須對應到父表的主鍵值** 或為 NULL。

範例

在「訂單資料表」中，`客戶ID (CustomerID)` 是外鍵，指向「客戶資料表」的主鍵：

```
CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(50)
);

CREATE TABLE Order (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
);
```

關聯說明：

- 每筆訂單的 `CustomerID` 必須是 `Customer` 表中已存在的客戶 ID。

---

3. 複合鍵（Composite Key）

定義

由 **兩個或多個欄位組合** 形成的主鍵，用於當單一欄位無法唯一識別資料時。

- 組合欄位值需唯一，但個別欄位可重複。

範例

在「選修課程表」中，`學生ID (StudentID)` 和 `課程ID (CourseID)` 共同組成複合主鍵：

```
CREATE TABLE Enrollment (
    StudentID INT,
    CourseID INT,
    Grade CHAR(2),
    PRIMARY KEY (StudentID, CourseID)
);
```

資料範例：

StudentID	CourseID	Grade
S001	C101	A

StudentID	CourseID	Grade
S001	C102	B
S002	C101	A

---

4. 候選鍵（Candidate Key）

定義

候選鍵是資料表中 **能唯一識別每筆資料的欄位或欄位組合**，且符合以下條件：

- **唯一性**：值不可重複。
- **最小性**：無法移除任一欄位仍保持唯一性。
- 主鍵是從候選鍵中選出的其中一個。

範例

在「學生資料表」中，`學號 (StudentID)` 和 `Email` 均為候選鍵：

```
CREATE TABLE Student (
    StudentID INT UNIQUE,
    Email VARCHAR(100) UNIQUE,
    Name VARCHAR(50)
);
```

候選鍵分析：

- `{StudentID}`：唯一且不可拆分。
- `{Email}`：唯一且不可拆分。
- `{StudentID, Email}`：符合唯一性，但包含多餘欄位，因此不是候選鍵。

鍵的對比表

鍵類型	定義	範例場景
主鍵	唯一識別資料的欄位	學生表的 `StudentID`
外鍵	指向另一表主鍵的欄位	訂單表的 `CustomerID`
複合鍵	多欄位組合的主鍵	選修表的 `(StudentID, CourseID)`
候選鍵	符合唯一性與最小性的欄位組合	學生表的 `Email`

實務應用建議

1. **主鍵設計**：優先使用單一欄位（如自動遞增 ID），避免複合鍵的複雜性。
2. **外鍵約束**：啟用資料庫的參照完整性檢查，防止無效關聯。
3. **候選鍵選擇**：若有多個候選鍵，選擇最簡潔且業務邏輯明確的作為主鍵。

關聯式資料庫的關聯類型

關聯式資料庫中的表格之間可以建立三種主要的關聯類型：一對一（1:1）、一對多（1:M）和多對多（M:N）。以下是每種類型的詳細說明與例子：

一對一關係（One-to-One, 1:1）

在一對一關係中，第一個表格中的每一筆記錄只能與第二個表格中的一筆記錄相關聯，反之亦然。

例子：

- 人員資料表與身分證資料表
  - 每個人只能有一張身分證
  - 每張身分證只能屬於一個人

資料表設計：

```
人員資料表：
- 人員ID (主鍵)
- 姓名
- 生日
- ...

身分證資料表：
- 身分證號碼 (主鍵)
- 人員ID (外鍵，參照人員資料表)
- 發證日期
- 有效期限
- ...
```

一對多關係（One-to-Many, 1:M）

在一對多關係中，第一個表格中的一筆記錄可以與第二個表格中的多筆記錄相關聯，但第二個表格中的每一筆記錄只能與第一個表格中的一筆記錄相關聯。

例子：

- 部門與員工
  - 一個部門可以有多名員工
  - 但一名員工只能屬於一個部門

資料表設計：

```
部門資料表：
- 部門ID（主鍵）
- 部門名稱
- 部門主管
- ...

員工資料表：
- 員工ID（主鍵）
- 員工姓名
- 部門ID（外鍵，參照部門資料表）
- 職稱
- ...
```

多對多關係（Many-to-Many, M:N）

在多對多關係中，第一個表格中的一筆記錄可以與第二個表格中的多筆記錄相關聯，反之亦然。多對多關係通常需要使用第三個關聯表格（junction table）來實現。

例子：

- 學生與課程
  - 一名學生可以選修多門課程
  - 一門課程可以被多名學生選修

資料表設計：

```
學生資料表：
- 學生ID（主鍵）
- 學生姓名
- ...

課程資料表：
- 課程ID（主鍵）
- 課程名稱
- 學分數
- ...

選課資料表（關聯表格）：
- 選課ID（主鍵）
- 學生ID（外鍵，參照學生資料表）
- 課程ID（外鍵，參照課程資料表）
- 成績
- ...
```

在多對多關係中，關聯表格（如上例的選課資料表）不僅僅連接兩個實體表格，還可以儲存關係本身的屬性（如成績）。關聯表格的主鍵通常是由兩個外鍵組成的複合主鍵，或者使用獨立的主鍵（如選課ID）。

這三種關係類型是設計關聯式資料庫結構的基本元素，正確地定義這些關係有助於確保資料的完整性、減少資料冗餘並提高查詢效率。

mandy0916473122@gmail.com