資料庫管理期末專題報告
Open Ticket 票務管理系統

Group 8
曾煥軒 B12705002、陳泊華 B12705014、張佑丞 B12705055

Github 連結　影片連結

1. 系統分析

1.1 市場需求與現有平台評估

近年來，人們對各類活動的需求不斷上升，活動主辦方對於輕量化、便捷、高擴展性的票務管理需求有逐漸上漲的趨勢。目前市面上有多個票務系統可供選擇，並根據活動種類的不同，主辦方和參與者對平台會有不同的要求與偏好。例如：在台灣，一部分的大型演唱會使用拓元訂票系統（如周杰倫巡迴演出），確保平台具備高流量乘載的穩定性；而展覽、演講則會選用 KKTIX / Accupass 等平台，這些平台在方便調整的同時，也能有效節省平台中介費用。

然而，我們發現，現有的訂票平台對於「小型活動舉辦方」而言並不是那麼方便、好用。首先，大多數訂票平台都會向活動主辦方收取一定比例的手續費，這對預算有限的小型活動來說，無疑增加了成本負擔。其次，這些平台的功能通常較為固定，缺乏靈活的客製化選項，使得主辦方難以根據活動的特性進行調整。最後，大部分平台在創建活動的使用界面較為繁瑣、耗時，對於每月甚至每週舉辦的活動（如讀書會、定期聚會）而言，管理起來更加困難。因此，我們決定開發 **Open Ticket** 來改善這些問題。

Open Ticket 是一個簡化活動管理的票務管理平台。用戶可以瀏覽、查詢自己喜歡的講座、比賽、音樂會等活動，參與並管理這些活動票卷，並盡量確保選位流程的正確性。而活動主辦方則可以控制每場活動的資訊，得到票務銷售情況並分析。我們也將這些程式開源出來，讓每個人都可以使用，也提供使用者自行擴展功能的可能性，以此達到輕量化、好用的目標。

1.2 目標使用者

1.2.1 用戶 (User)

- 描述：希望參加活動並購買票券的個人或團體的一般用戶。
- 需求：
    - 瀏覽各類活動資訊。
    - 搜尋和篩選活動。
    - 購買票券並選擇座位。

- 查看和管理訂單歷史。
- 接收活動更新。

1.2.2 主辦方 (Organizer)

- 描述:負責策劃並管理活動的組織或個人。
- 需求:
  - 創建和發布活動資訊。
  - 管理票種、價格及數量。
  - 監控票務銷售狀況。
  - 查看參加者名單及聯絡資訊。

1.2.3 系統管理員 (Admin)

- 描述:負責監控及管理平台的整體運營。
- 需求:
  - 管理用戶帳號和權限。
  - 監控系統性能和安全性。
  - 管理和審核活動內容。

1.3 主要功能實現

1.3.1 用戶功能

1. 註冊與登入
   - 註冊帳號，創建使用者在資料庫中。
   - 根據使用者的身份驗證訊息登入，得到身分 token 已執行後續功能。
2. 活動瀏覽與搜尋
   - 瀏覽不同類型的活動（如音樂會、講座、比賽等），並獲得演出者、日期、時間、地點及票價資訊。。
   - 使用關鍵字、地點、日期等條件查詢特定活動的票務狀況，查看可用座位與價格。
3. 票券購買與選擇
   - 選擇票種、數量並進行購買。
   - 如果活動有座位選擇，提供互動式選位界面。
   - 支援線上付款支付（信用卡、行動支付等）。
4. 訂單管理
   - 查看訂單歷史和狀態。
   - 取消訂單或申請退款。

1.3.2 主辦方功能

1. 活動創建與管理
   - 輕鬆創建新活動，填寫活動名稱、描述、日期、地點等基本資訊。
2. 票務管理
   - 設定票種（如一般票、VIP票等）、價格及數量。
   - 管理票券的銷售狀態（開售、停售）。
   - 設定促銷活動和折扣碼。

3. 參加者管理
    - 查看購票者名單及聯絡資訊。
    - 發送活動通知或重要資訊給參加者。

1.3.3 系統管理員功能

1. 用戶與主辦方管理
    - 審核和管理用戶註冊申請。
    - 設定和調整用戶權限(如升級為主辦方)。
2. 平台監控
    - 監控系統性能、流量和資源使用情況。
    - 檢測並處理安全威脅(如DDoS攻擊、數據洩露等)。
3. 內容審核
    - 審核主辦方發布的活動內容，確保符合平台政策。
    - 處理用戶和主辦方的投訴和報告。

1.4 非功能需求

1.4.1 性能需求

- 高可用性:確保平台在高流量時段仍能穩定運行，避免系統崩潰或響應遲緩。
- 快速響應:用戶操作(如瀏覽、購票、管理活動等)應在合理時間內完成，提升使用體驗。並用 Index 加速慣用操作的速度。

1.4.2 可擴展性

- 模組化設計:平台應具備良好的模組化結構，方便未來功能擴展和維護。

1.4.3 安全性

- 資料保護:加密用戶敏感資料(如個人資訊、支付信息)，防止未經授權的存取。
- 權限控制:嚴格管理不同角色的訪問權限，確保各類用戶只能訪問其應有的功能和數據。

2. 系統設計

Open Ticket 票務系統的設計基於活動訂票的核心需求，涵蓋用戶、活動、場地、座位、票券及訂單的管理。以下將詳細說明系統的架構與設計。

2.1 ER Diagram

圖1是 Open Ticket 的 ER Diagram，在這個ERD中主要有以下七個實體（Entity）以及一些關係（Relationship），最終產生了 8 張 Table，簡要說明如下：

1. **USER (用戶)**：USER 代表的是使用 Open Ticket 票務系統的任何人，任何人都須註冊才能開始使用。註冊時系統會要求使用者提供名稱、信箱、密碼，經註冊後便會產生一個專屬於該位使用者的編號，並定義他在 Open Ticket 中的身份為 User。經過後臺手動設定，可以將特定使用者的身份修改為 Organizer 或 Admin，此時該使用者便可以使用相對應功能（如第一章所述）。
2. **EVENT (活動)**：每場活動由主辦方建立，記錄活動的基本資訊（名稱、演出者、日期、地點等）。每場活動只能在一個場地舉行，且可包含多種類型的票券，以適應不同參加者需求。
3. **VENUE (場地)**：定義活動舉行的地點。場地包含一組唯一的座位配置（SEAT），並具有基本屬性如地點名稱、地址、容量等。多場活動可以在同一場地舉行，但每場活動僅關聯一個場地。
4. **SEAT (座位)**：定義場地內的座位配置，包括區域、列、座位號及座位類型。座位信息與票券信息緊密相關，以便為每場活動的座位分配具體的票券。
5. **TICKET (票券)**：每張票券關聯一個座位和活動，用戶在選購票券時會生成票券編號。票券類型和價格由主辦方定義，以便於不同座位需求的分配。
6. **ORDER (訂單)**：記錄用戶的每筆交易細節，包括訂單金額、訂購日期、訂單狀態（「待付款」、「已付款」等）。訂單一旦確認，會建立票券的購買關係。
7. **PAYMENT (付款)**：訂單生成後，使用者會進行付款，並記錄付款的金額、方式（信用卡、銀行轉帳等）以及付款狀態，確保交易的可追溯性。
8. **ORGANIZE**：主辦方與活動之間的關係，表示主辦方所策劃的活動。

使用者能夠查詢活動列表，針對有興趣的活動點擊查看更詳細活動資料、查詢該活動座位使用情形；系統也會根據活動舉辦時間進行狀態處理。使用者可在對應的活動找到對應的場地和座位資訊，並且可以選擇一至多個位子進行訂票（詳細流程邏輯將於「3.6 併行控制」說明）。按下確認選位後，系統會產生對應訂單並進入付款頁面，此時使用者可輸入相關資訊完成付款。

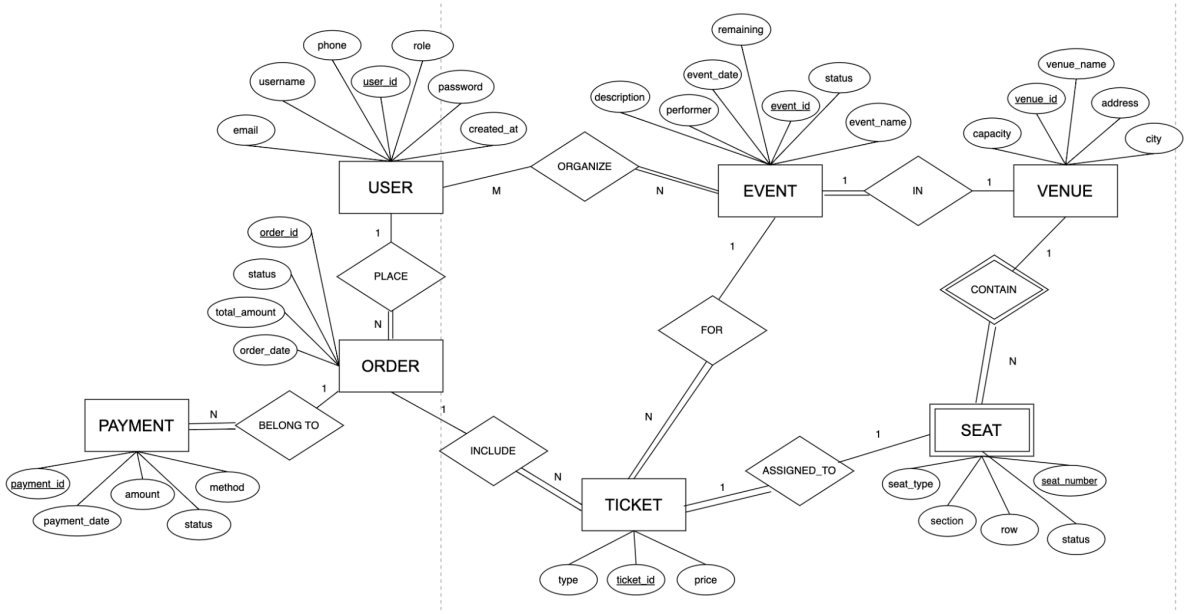


圖1: Open Ticket ERD

2.2 Relational Database Schema Diagram

根據上述 ER Diagram，我們可建立以下資料庫架構圖（Database Schema Diagram），一共有八個關聯（relation），分別是USER、ORGANIZE、EVENT、VENUE、SEAT、TICKET、ORDER、PAYMENT：

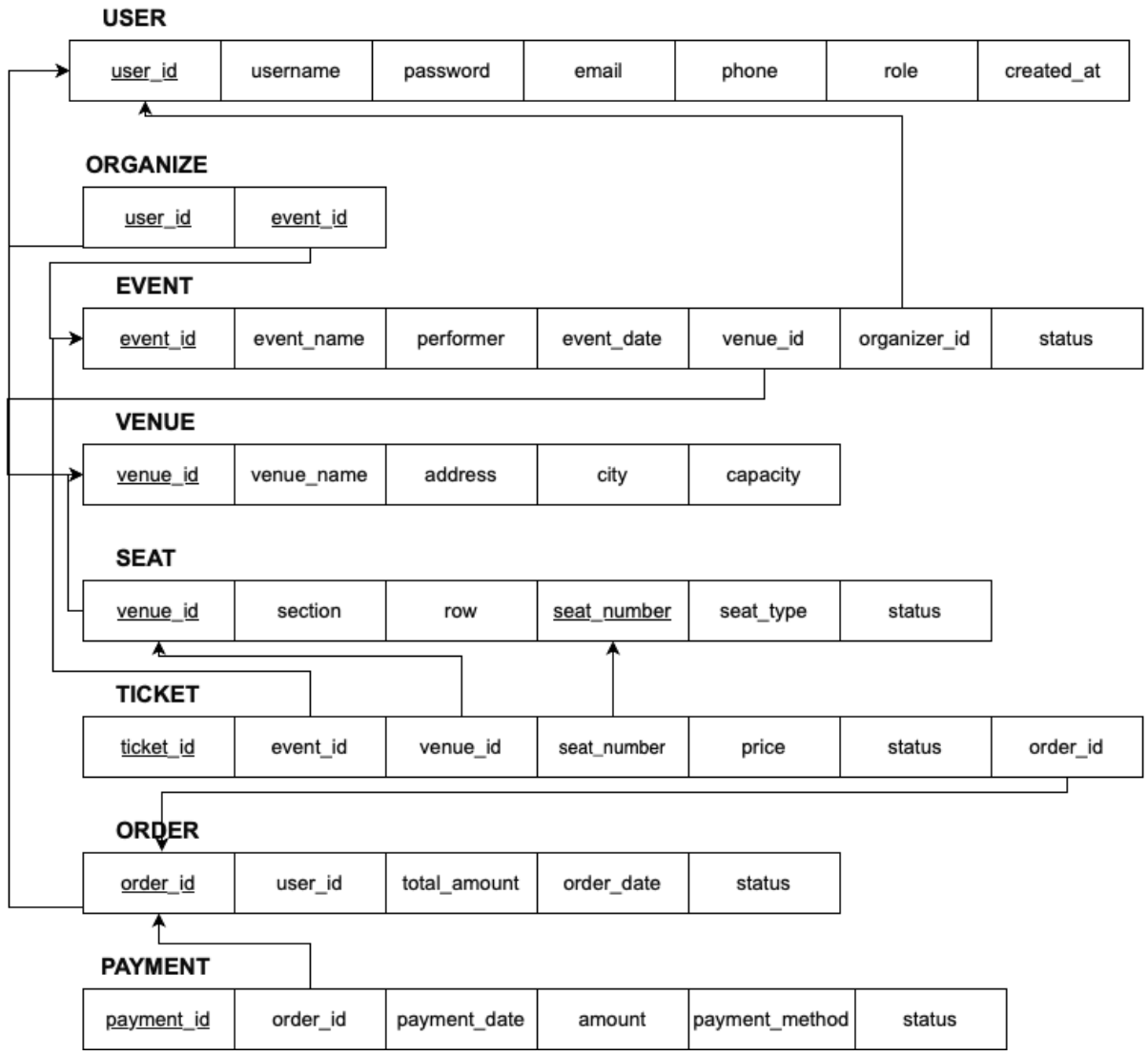


圖 2:「Open Ticket」的 Relational Database Schema Diagram

USER 關聯的主鍵是 user_id，該屬性被設置為系統自動遞增（Serial）唯一識別碼，用於標識每個用戶。該表存儲用戶的基本信息，包括用戶名 (username)、密碼 (password)、電子郵件 (email)、電話號碼 (phone)、角色 (role)，以及帳戶創建時間 (created_at)。由於 role 是多值屬性，因此被分離至 USER_ROLE 表中。PARTICIPATE 關聯表則用於描述用戶與活動之間的多對多關係，其主鍵由 user_id 和 event_id 組成，分別參考 USER 表的主鍵和 EVENT 表的主鍵，用於記錄用戶參與的活動但不包括活動的創建者。同樣地，ORGANIZE 表記錄了用戶作為主辦方組織活動的關係，其主鍵也由 user_id 和 event_id 組成，分別參考 USER 表和 EVENT表的主鍵。

EVENT 表的主鍵是 event_id，被設置為系統自動遞增的唯一識別碼，用於標識每場活動。該表存儲了活動的基本信息，包括活動名稱 (event_name)、表演者 (performer)、活動日期 (event_date)、場地 ID (venue_id，參考 VENUE 表的主鍵)、主辦方 ID (organizer_id，參考 USER 表的主鍵) 和活動的狀態 (status)。此外，場地的相關信息存儲在 VENUE 表中，其主鍵是 venue_id，用於唯一標識場地。該表記錄了場地的名稱 (venue_name)、地址 (address)、城市 (city) 和容納人數上限 (capacity)。

座位的詳細信息存儲在 SEAT 表中，其組合主鍵由 venue_id 和 seat_number 組成，確保每個座位在特定場地中是唯一的。SEAT表還包括區域 (section)、排號 (row)、座位類型 (seat_type，如 Regular 或 VIP) 和狀態 (status，如 Available 或 Sold)。而票券的信息存儲在 TICKET 表中，其主鍵為 ticket_id，該表參考了 EVENT 表的主鍵 event_id 和 SEAT 表的座位組合鍵，用於記錄每張票的詳細信息，包括票價 (price) 和狀態 (status)。

訂單信息存儲在 ORDER 表中，其主鍵是 order_id，記錄了用戶 (user_id，參考 USER 表) 所下訂單的總金額 (total_amount)、訂單日期 (order_date) 和狀態 (status)。最後，PAYMENT 表記錄了與訂單相關的付款信息，其主鍵是 payment_id，包括付款金額 (amount)、付款日期 (payment_date)、付款方式 (payment_method) 和狀態 (status)。這些關聯表共同組成了整個系統的資料庫架構，確保能有效地管理用戶、活動、場地、座位、票券、訂單和支付信息。

2.3 Data Dictionary

「Open Ticket」的資料表共有圖 3 所示的 8 張，各個資料表的欄位相關資訊依序呈現在表 1 到表 7。

1. USER

Column Name	Meaning	Data Type	Key Constraint	Domain
user_id	使用者代號	BIGINT	PK, Not Null	
username	使用者名稱	VARCHAR(50)	Not Null, Unique	
password	使用者密碼	VARCHAR(100)	Not Null	
email	電子郵件	VARCHAR(100)	Not Null, Unique	
phone	聯絡電話	VARCHAR(20)	Null	
role	用戶角色	VARCHAR(20)	Not Null	{User, Organizer, Admin}
created_at	帳號建立時間	DATETIME	Not Null	

2. EVENT

Column Name	Meaning	Data Type	Key Constraint	Domain
event_id	演唱會代號	BIGINT	PK, Not Null	
event_name	演唱會名稱	VARCHAR(100)	Not Null	
performer	演出者	VARCHAR(50)	Not Null	
event_date	演唱會日期	DATE	Not Null	
venue_id	場地代號	INT8	FK: VENUE(venue_id)	
organizer_id	主辦方代號	BIGINT	FK: USER(user_id)	
status	活動狀態	VARCHAR(20)	Default: ‘Scheduled’	{Scheduled, Canceled, Completed}

3. VENUE

Column Name	Meaning	Data Type	Key Constraint	Domain
venue_id	場地代號	BIGINT	PK, Not Null	
venue_name	場地名稱	VARCHAR(50)	Not Null	
address	地址	VARCHAR(200)	Not Null	
city	城市	VARCHAR(50)	Not Null	
capacity	容納人數	INT	Not Null	
status	借用狀態	VARCHAR(20)		{Available, Reserved, Maintaining}

4. SEAT

Column Name	Meaning	Data Type	Key Constraint	Domain
venue_id	場地代號	INT8	FK: VENUE(venue_id)	
seat_number	座位號碼	VARCHAR(5)	Unique (venue_id, seat_number)	
section	區域	VARCHAR(20)		
row	列	VARCHAR(5)		
seat_type	座位類型	VARCHAR(20)		{Regular, VIP, Wheelchair}
status	座位狀態	VARCHAR(20)	Default: ‘Available’	{Sold, Reserved, Available}

5. TICKET

Column Name	Meaning	Data Type	Key Constraint	Domain
ticket_id	票券代號	BIGINT	PK, Not Null	
event_id	演唱會代號	BIGINT	FK: EVENT(event_id)	
seat_id	座位代號	BIGINT	FK: SEAT(seat_id)	
price	票價	DECIMAL(10, 2)	Not Null	
type	票券類型	VARCHAR(20)	Default: ‘Adult’	{"Youth", "Adult", "Senior"}

6. ORDER

Column Name	Meaning	Data Type	Key Constraint	Domain
order_id	訂單代號	BIGINT	PK, Not Null	
user_id	使用者代號	BIGINT	FK: USER(user_id)	
total_amount	訂單總金額	DECIMAL(10, 2)	Not Null	
order_date	訂單日期	DATETIME		
status	訂單狀態	VARCHAR(20)	Default: ‘Pending’	{Pending, Paid, Canceled}

7. PAYMENT

Column Name	Meaning	Data Type	Key Constraint	Domain
payment_id	付款代號	BIGINT	PK, Not Null	
order_id	訂單代號	BIGINT	FK: ORDER(order_id)	
payment_date	付款日期	DATETIME		
amount	付款金額	DECIMAL(10, 2)	Not Null	
method	付款方式	VARCHAR(20)	Not Null	{Credit Card, PayPal, Bank Transfer}
status	付款狀態	VARCHAR(20)		{Pending, Completed, Failed}

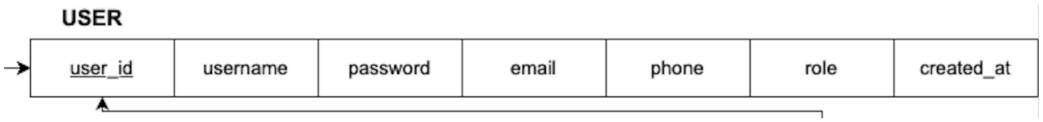
8. ORGANIZE

Column Name	Meaning	Data Type	Key Constraint	Domain

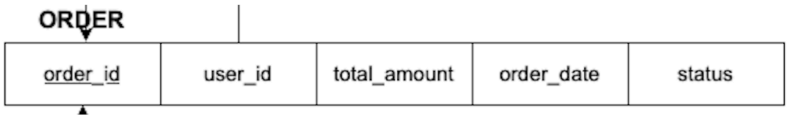
2.4 正規化分析

為保證數據一致性和結構最佳化，Open Ticket 票務系統於設計時遵循正規化（Normalization）原則，而此可透過檢視資料庫綱目（database schema）來檢驗。我們將從第一正規式（1NF）至第四正規式（4NF）分別說明 Open Ticket 如何滿足這些規則。

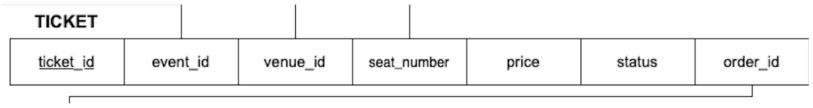
1. **1NF**：確保每個關聯的所有欄位皆為單一值且不可分割。Open Ticket 的各資料表中均已排除重複值或多值欄位（multivalue），例如 USER 表中的 `username` 和 `email` 為唯一值且單一。



2. **2NF**：在符合 1NF 的前提下，Open Ticket 確保每個非鍵屬性都完全功能相依（fully functional dependency）於任一候選鍵（candidate key）。舉例而言，ORDER 表中的 `total_amount` 和 `order_date` 僅依賴於 `order_id`，避免了部分依賴性。



3. **3NF**：在符合 2NF 的前提下，所有非鍵屬性不應依賴（transitive dependency）於主鍵之外的其他屬性。Open Ticket 各資料表的非鍵屬性都完全依賴於主鍵，例如 TICKET 表中 `price` 與 `event_id` 無間接依賴性，符合 3NF。



4. **BCNF**：Open Ticket 的資料表進一步要求每個功能依賴的左側為超鍵（superkey），保證所有候選鍵在設計上都符合 BCNF。
5. **4NF**：由於「Open Ticket」的所有關聯都不存在多值相依(multi-valued dependency)，因此滿足 4NF。

3 系統實作

3.1 資料庫建置方式及資料來源說明

關於 USER 和 EVENT 資料表內的資料建置方式，我們首先利用隨機生成的方法建立了 20,000 筆使用者資料，這些資料包括使用者的基本資訊，如使用者名稱、密碼、電子郵件、電話號碼以及角色。生成的使用者中，部分被指定為主辦方 (Organizer)，其餘則為一般使用者 (User)。隨後，模擬這些主辦方創建活動的行為，隨機選擇活動名稱、表演者以及場地等相關資訊，共生成 13,000 筆活動資料，並儲存於 EVENT 資料表中。此外，為每場活動分配隨機的場地和活動日期，並確保每場活動的時間與場地不發生衝突。

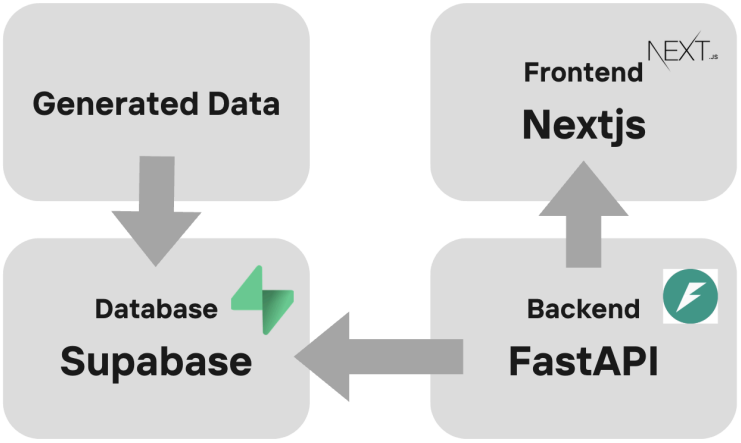
接下來，我們利用已經生成的活動資料，為每場活動分配座位，這些資料儲存於 SEAT 資料表中。每個場地依據其容納人數的上限生成對應的座位數，並對每個座位指定區域 (Section)、排號 (Row)、座位類型 (如 Regular 或 VIP 或 Wheelchair) 和初始狀態 (Available)。最終生成了超過 60,000 筆座位資料。

針對 ORDER 和 TICKET 資料表，我們模擬用戶下訂單的行為，隨機選擇用戶和活動進行購票操作，並為每筆訂單生成唯一的訂單編號、訂購日期和總金額。每筆訂單包含若干張票券資料，票券的資訊包括票券所對應的活動、座位、價格和狀態 (如 Sold 或 Reserved)。最終生成了 20,000 筆訂單資料和 20,000 筆票券資料，分別儲存在 ORDER 和 TICKET 資料表中。

最後，PAYMENT 資料表則記錄了用戶的付款行為。每筆付款資料與對應的訂單相關聯，包含付款日期、付款金額、付款方式 (如信用卡或行動支付) 和付款狀態 (如 Completed 或 Pending)。模擬用戶的多樣化支付行為，共生成了 15,000 筆付款記錄，完整地反映了系統的交易流程。

3.2 實作架構

根據前述資料庫建置說明，此處附上整體專案架構供參考，其中supabase為基於postgreSQL的一個雲端資料庫系統。



3.3 重要功能及對應的 SQL 指令

第 1.1 節中我們有介紹一些給 User 及 Organizer 的功能。在第 3.3.1 和第 3.3.2 小節中將列出附帶特定情境下，完成這些功能所使用的(一或數個)SQL 指令。此外，在第 3.3.3 小節中，我 們也列出系統級指令其對應的 SQL 指令。

3.3.1 給 User 的功能

1. 建立活動 (Create Event)
若要實現此功能，假設情境為「主辦方 Organizer_id『1024』欲創建一場名為『音樂盛典』的活動，活動表演者 Performer 是『John Doe』，活動地點 Venue_id 為『5』，活動狀態 Status 設為『Scheduled』，活動日期 Event_date 是『2024-12-25』，活動描述 Description 是『年度最盛大的音樂會』。」系統會在 EVENT 資料表中新增一筆活動資料，並自動生成唯一的 Event_id。
對應的 **SQL** 指令如下：

```
Insert Into EVENT (event_name, performer, event_date, venue_id,
```

```
organizer_id, description, status)
Values ('音樂盛典', 'John Doe', '2024-12-25', 5, 1024, '年度最盛大的
音樂會', 'Scheduled');
```

2. 查詢可預訂的座位 **(Query Available Seats)**
若要實現此功能，假設情境為「查詢活動編號 Event_id 為『2048』且座位狀態 Status 為『Available』的所有座位資訊。」系統會根據條件篩選 SEAT 資料表中符合條件的記錄，並回傳所有可供預訂的座位資訊。
對應的 **SQL** 指令如下：

```
Select *
From SEAT
Where event_id = 2048 And status = 'Available';
```

3. 購買票券 **(Purchase Tickets)**
若要實現此功能，假設情境為「使用者 User_id『1503』欲購買活動編號 Event_id『2048』中座位號碼為 101 和 102 的票券，每張票券價格為 500，訂單應自動生成唯一的 Order_id，訂單狀態設為『Pending』，並記錄訂單日期 Order_date 為『2024-12-01』。」系統會執行以下操作：

1. 在 ORDER 資料表中新增一筆訂單資料。
2. 在 TICKET 資料表中新增兩筆票券資料，分別記錄對應的座位資訊。

對應的 **SQL** 指令如下：

```
-- 建立訂單
Insert Into `ORDER` (user_id, total_amount, order_date, status)
Values (1503, 1000, '2024-12-01', 'Pending');
-- 獲取自動生成的 Order_id
Set @newOrderId = LAST_INSERT_ID();
-- 為訂單新增票券
Insert Into TICKET (event_id, venue_id, seat_number, price,
status, order_id)
Values
(2048, 5, 101, 500, 'Sold', @newOrderId),
(2048, 5, 102, 500, 'Sold', @newOrderId);
```

4. 查詢付款狀態 **(Query Payment Status)**
若要實現此功能，假設情境為「查詢訂單編號 Order_id 為『3078』的付款狀態。」系統會在 PAYMENT 資料表中篩選並回傳對應的付款記錄資訊。
對應的 **SQL** 指令如下：

```
Select status
From PAYMENT
Where order_id = 3078;
```

3.3.2 給 Organizer 的功能

1. 創建活動
功能描述: Organizer 可以在系統中創建一個新活動，指定活動名稱、表演者、日期、場地、描述和狀態等詳細資訊。
對應 **SQL** 指令:

```
Insert Into EVENT (event_name, performer, event_date, venue_id,
organizer_id, description, status)
Values ('活動名稱', '表演者', '2024-12-31', 1, 1024, '活動描述',
'Scheduled');
```

2. 更新活動資訊
功能描述: Organizer 可以更新其已創建活動的資訊(例如更改日期、狀態或描述等)。
對應 **SQL** 指令:

```
Update EVENT
Set event_name = '新活動名稱',
    performer = '新表演者',
    event_date = '2024-12-25',
    description = '更新後的描述',
    status = 'Scheduled'
Where event_id = 2001 And organizer_id = 1024;
```

3. 查看自己創建的所有活動
功能描述: Organizer 可以查詢所有自己創建的活動，包括活動名稱、日期、狀態等資訊。
對應 **SQL** 指令:

```
Select *
From EVENT
Where organizer_id = 1024;
```

4. 查詢活動的參與者清單
功能描述: Organizer 可以查詢某個活動的所有參與者，包括參與者的用戶名稱和聯絡方式。
對應 **SQL** 指令:

```
Select u.username, u.email, u.phone
From PARTICIPATE As p
Join USER As u On p.user_id = u.user_id
Where p.event_id = 2001;
```

5. 查詢活動的票券銷售情況

功能描述: Organizer 可以查看某活動的票券銷售統計，包括售出票券數量及總收入。
對應 **SQL** 指令:

```
Select Count(*) As sold_tickets, Sum(price) As total_revenue
From TICKET
Where event_id = 2001 And status = 'Sold';
```

6. 管理活動的座位
功能描述: Organizer 可以對活動場地的座位進行管理，例如查詢哪些座位已被預訂或仍可用。
對應 **SQL** 指令:

```
-- 查詢可用座位
Select *
From SEAT
Where event_id = 2001 And status = 'Available';
-- 查詢已售座位
Select *
From SEAT
Where event_id = 2001 And status = 'Sold';
```

7. 分析活動表現
功能描述: Organizer 可以分析某活動的表現，例如座位利用率、總參與人數和收入。
對應 **SQL** 指令:

```
Select
    Count(*) As total_seats,
    Sum(Case When status = 'Sold' Then 1 Else 0 End) As
utilized_seats,
    Round(Sum(Case When status = 'Sold' Then 1 Else 0 End) * 100.0
/ Count(*), 2) Asseat_utilization_rate,
    Sum(Case When status = 'Sold' Then price Else 0 End) As
total_revenue
From SEAT
Where event_id = 2001;
```

8. 查看近期訂單
功能描述: Organizer 可以查看最近幾筆訂單的詳細資訊，包括訂單金額、日期及狀態。
對應 **SQL** 指令:

```
Select o.order_id, o.total_amount, o.order_date, o.status
From `ORDER` As o
Join EVENT As e On o.event_id = e.event_id
```

```
Where e.organizer_id = 1024
Order By o.order_date Desc
Limit 10;
```

9. 查詢某活動的預訂狀態
功能描述: Organizer 可以查詢某活動中每個座位的當前狀態，包括是否可預訂或已售出。
對應 **SQL** 指令:

```
Select section, row, seat_number, seat_type, status
From SEAT
Where event_id = 2001
Order By row, seat_number;
```

3.2.1 給 Admin 的功能

1. 查詢所有活動
功能描述: Admin 可以查詢所有活動，包括狀態、主辦方和場地等信息。
對應 **SQL** 指令:

```
Select e.event_id, e.event_name, e.event_date, v.venue_name,
u.username As organizer, e.status
From EVENT As e
Join VENUE As v On e.venue_id = v.venue_id
Join USER As u On e.organizer_id = u.user_id;
```

2. 查詢票券銷售記錄
功能描述: Admin 可以查看所有票券的銷售數據，包括已售出數量及未售出座位。
對應 **SQL** 指令:

```
Select t.ticket_id, t.event_id, t.seat_number, t.price, t.status
From TICKET As t
Order By t.event_id, t.seat_number;
```

3.4 SQL 指令效能優化與索引建立分析

在 **OpenTicket** 平台中，為了確保資料庫操作的高效能，我們對常用的 SQL 查詢進行效能優化，並根據查詢需求建立適當的索引。以下將針對具體的 SQL 指令進行效能優化與索引建立的分析，並使用 `EXPLAIN ANALYZE` 來評估查詢效能的改進。

根據 **OpenTicket** 平台的查詢需求，我們制定了以下索引建立策略:

1. 高頻查詢欄位建立索引：
    - 針對經常用於查詢條件（如 `WHERE` 子句）的欄位，如 `event_date`、`venue_id` 等，建立相應的索引以提升查詢效率。
2. 聯接操作相關欄位建立索引：
    - 對於需要進行表間聯接的欄位，如 `seats.venue_id` 和 `venues.venue_id`，建立索引以加速聯接操作。

範例一：事件日期索引

```
CREATE INDEX event_date ON events(event_date);
```

```
EXPLAIN ANALYZE
SELECT *
FROM events
WHERE event_date > '2024-10-01';
```

分析與效能提升：

建立索引：

1. 目的：為 `events` 表中的 `event_date` 欄位建立索引，以加速基於日期篩選的查詢。
2. 效果：建立索引後，資料庫在執行 `WHERE event_date > '2024-10-01'` 的查詢時，能夠快速定位符合條件的記錄，避免全表掃描。

範例二：場地**ID**索引

```
CREATE INDEX idx_seats_venue_id ON seats (venue_id);
```

```
EXPLAIN ANALYZE
SELECT *
FROM seats
WHERE venue_id = 5;


EXPLAIN ANALYZE
SELECT *
FROM seats
INNER JOIN venues
ON seats.venue_id = venues.venue_id
```

```
    WHERE seats.venue_id = 5;
```

分析與效能提升：

建立索引：

1. 目的：為 `seats` 表中的 `venue_id` 欄位建立索引，以加速基於場地ID的查詢。
2. 效果：建立索引後，資料庫在執行 `WHERE venue_id = 5` 的查詢時，能夠快速定位符合條件的記錄，提升查詢效率。

查詢一：單表查詢

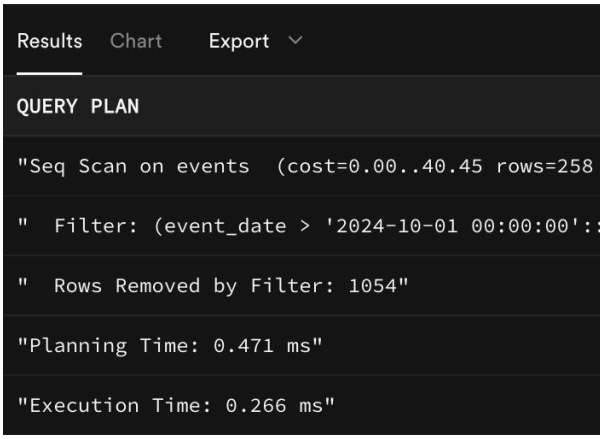
```
EXPLAIN ANALYZE
SELECT *
FROM seats
WHERE venue_id = 5;
```

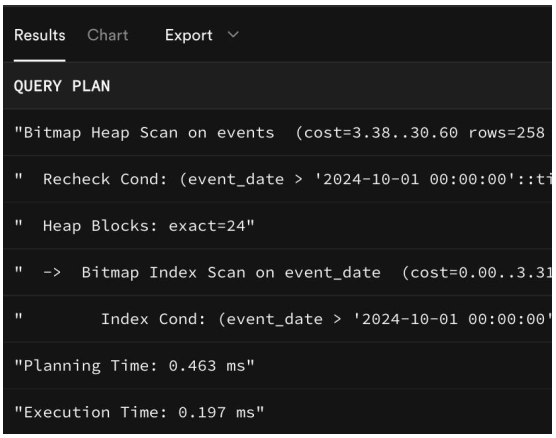
查詢二：多表聯接查詢

```
EXPLAIN ANALYZE
SELECT *
FROM seats
INNER JOIN venues
ON seats.venue_id = venues.venue_id
WHERE seats.venue_id = 5;
```

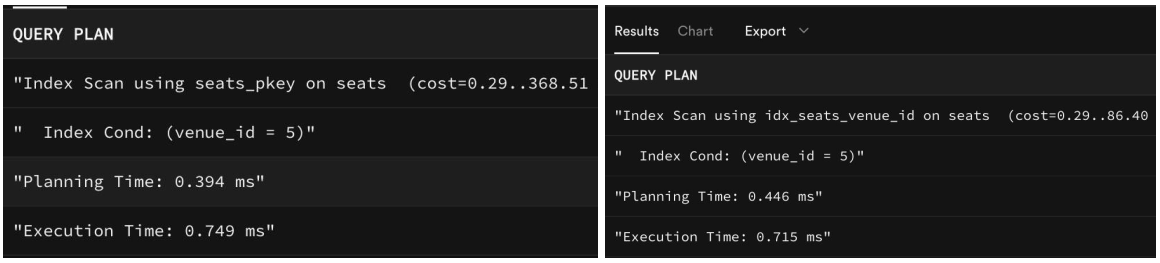
以下為建立索引前後的查詢效能測試結果示例：

查詢 **1:** 節省 **25%** 查詢活動花費

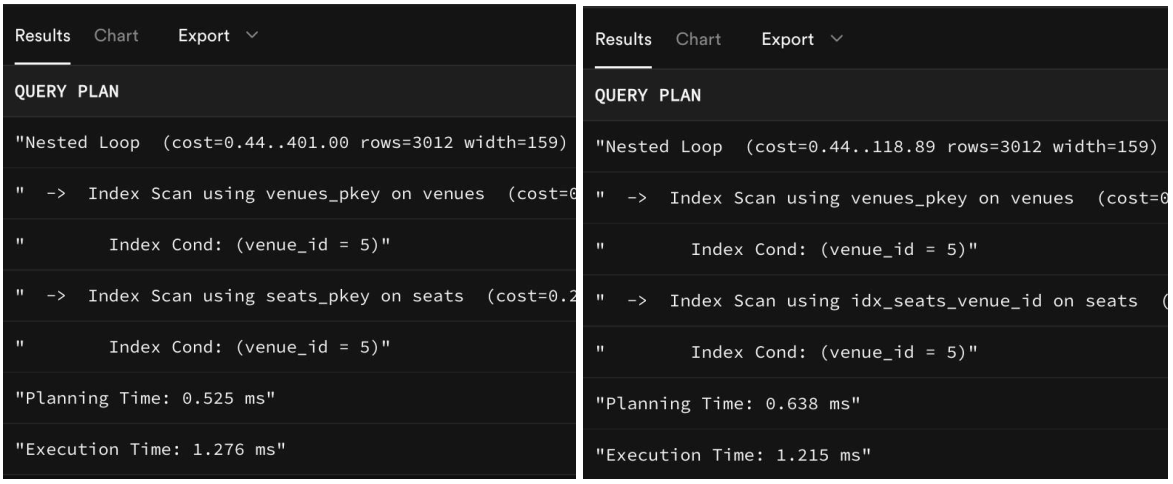




查詢 **2-1:** 節省 **70%** 查詢活動座位花費



查詢 **2-2** 節省 **76%** 查詢活動座位花費



總結:透過建立適當的索引，查詢座位與查詢活動兩個操作的 cost 顯著降低，但因為目前資料量不大，執行時間不長，所以 Planning Time 反而因為 query planning 的選擇增加而增加。以下是索引帶來的改變(左圖為為加 Index，右圖為加 Index)

3.5 交易管理

在管理票務的部分，我們提供了一個批次創建票券的功能。該功能允許主辦方一次性創建多個票券，並確保在創建過程中數據的一致性和完整性。在 `./app/crup` 目錄下的 `create_tickets` 函式中，從客戶端接收一組票券資料後，系統會將這些資料轉換為 `Ticket` 實例，並透過資料庫會話(`Session`)將這些票券批次新增至資料庫中。

以下是 `create_tickets` 函式的實現範例:

```
from typing import List
from sqlalchemy.orm import Session
from .models import Ticket, TicketCreate

def create_tickets(db: Session, tickets: List[TicketCreate]) ->
List[Ticket]:
    db_tickets = [
```

```
        Ticket(
            event_id=ticket.event_id,
            order_id=ticket.order_id,
            venue_id=ticket.venue_id,
            seat_number=ticket.seat_number,
            price=ticket.price,
            type=ticket.type
        )
        for ticket in tickets  # 遍歷 tickets 列表中的每一個票券資料
    ]
    db.add_all(db_tickets)  # 使用 add_all() 方法批次新增票券
    db.commit()  # 提交交易，將所有票券資料寫入資料庫
    return db_tickets
```

功能說明

1. 資料接收與轉換：
    - 主辦方透過 API 發送包含多筆票券資訊的請求。
    - `create_tickets` 函式接收這些資料，並將每筆票券資料轉換為 `Ticket` 模型的實例。
2. 批次新增票券：
    - 使用 `db.add_all(db_tickets)` 方法，將所有 `Ticket` 實例一次性添加至資料庫會話中。這樣可以減少與資料庫的交互次數，提高效率。
3. 提交交易：
    - 呼叫 `db.commit()` 方法，提交交易以將所有新增的票券資料寫入資料庫。
    - SQLAlchemy 的 `Session` 在預設情況下會自動管理交易。當呼叫 `commit()` 時，會自動開始一個交易，並在所有操作成功後提交交易。

交易處理機制

- 自動交易管理：
    - SQLAlchemy 的 `Session` 在執行 `add_all()` 和 `commit()` 方法時，會自動處理交易的開始與提交，無需手動執行 `BEGIN` 等交易控制操作。

- 錯誤處理與回滾：
    - 在提交交易的過程中，如果任何一筆資料因違反資料表的約束條件（例如，票券價格為負值、座位號碼重複等）而導致 `INSERT` 操作失敗，`db.commit()` 將會拋出例外（Exception）。
    - 當例外發生時，SQLAlchemy 會自動回滾（`rollback`）當前交易，撤銷所有在該交易中進行的資料庫變更，確保資料庫保持一致性。
    - 例如，若在批次新增票券的過程中，第三筆票券的資料不符合約束條件，系統

將停止新增操作，並撤銷前兩筆已新增的票券資料。

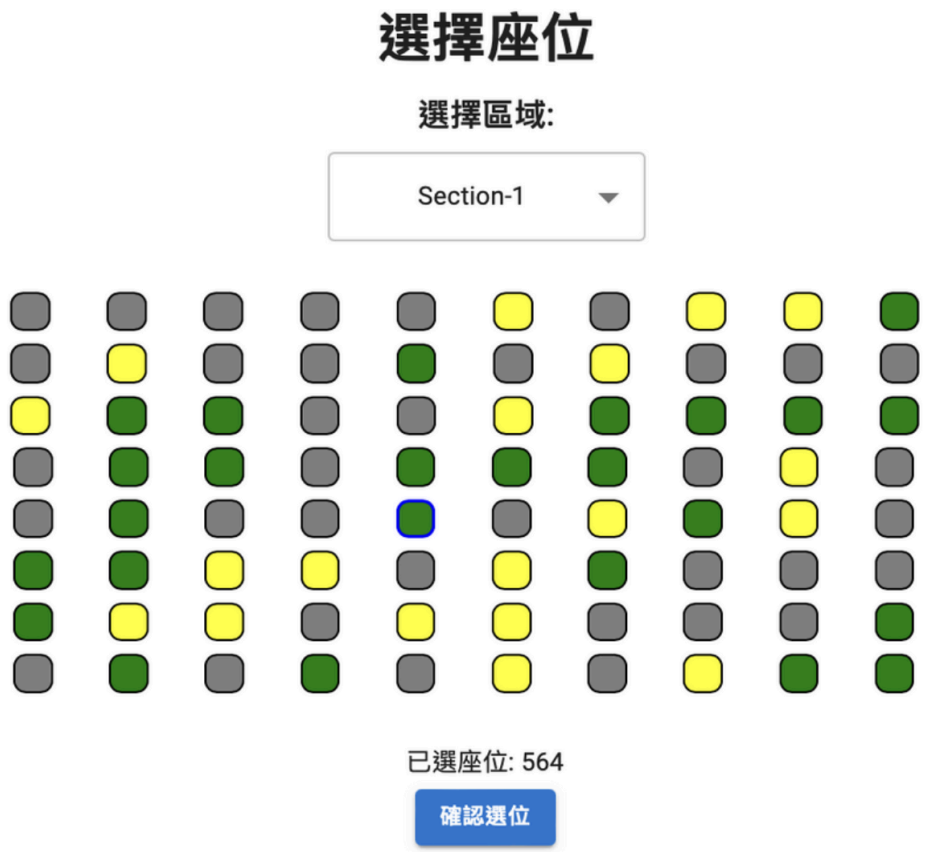
3.6 併行控制

在本系統中，當有多個使用者同時欲訂購同個座位便有加鎖的必要，確保一個座位只賣給至多一位購買者。資料庫層級部分，由於使用Postgre為基礎的supabase，在此方便已有併行控制處理；然為確保系統正常運作、使用者良好體驗並減少後端負擔，我們於前端亦實作了相關併行控制操作：

使用者選擇有興趣的活動查看剩餘座位，其中座位有三種狀態（見下圖）：灰色表「已售出」（Sold），代表選擇該座位的使用者已成功付款；黃色表「已預定」（Reserved），代表選擇該座位的使用者仍在付款頁面或付款仍在審核狀態；綠色則表該座位「可購買」（Available）。使用者可選擇一至多個座位，點擊「確認選位」後，確認該座位狀態確實為隨即至資料庫更新座位狀態為Reserved並進入付款頁面，模擬lock行為；若有其他使用者於此時查詢該活動座位，則無法選購。此時有三種可能情形：

1. 若使用者停留於付款頁面太久，則提示跳出回到首頁，並將座位狀態改回Available
2. 若點選「取消付款」，則將座位狀態改回Available並回到首頁
3. 若點選「確認付款」，則產生訂單（本系統沒有實際串接銀行API，所以確認付款後會直接將座位狀態更新為Sold）

如此可達到併行控制，確保正確購票行為。



4 分工資訊

陳泊華:

1. Payment Concurrency Control 相關前後端功能頁面實作
2. Seat status update, order & ticket create, Event list 等功能實作
3. 前端環境、頁面架設
4. Demo 影片、簡報製作

張佑丞:

1. 生成大量假資料
2. 實作數個後端api、設計organizer和admin分別所需要的分析功能
3. 前端優化使用者 UIUX體驗、實作create event、個人資料、活動管理者介面、admin介面
4. Demo影片、簡報製作

曾煥軒:

1. 後端架構建立，包含 ORM creation，DB connection，authentication
2. 實作大量 CRUD
3. 實作數個後端 router
4. Demo影片、簡報製作

5 專案心得

陳泊華:
很開心藉此機會紮實地走過從設計到實作的全端系統設計流程，也深刻體會到資料庫的設計對整個系統的影響之深。謝謝超給力的組員們，儘管各種版控之亂歷歷在目，但我相信這樣的溝通和協作是非常難能可貴的經驗，debug能力也是。希望未來有一天我們能真的讓這個系統串接銀行付款功能上線！我們嘎嘎頂

張佑丞:
關於這們課的專題實作從加簽到就開始期待了，也透過這們課了解到一個複雜的系統需要如何有條理地規劃其資料庫架構所需要考慮的種種因素，從發想題目到最後真的寫出一個嘎嘎頂的網站真的讓我非常有成就感。最後也感謝超級給力的組員們給我這個機會藉機學習前後端的架設與整合，雖然常常在奇怪的地方產生奇怪的錯誤XD，但相信這些經驗都能成為我未來學習的養分！資料庫衝衝衝！！

曾煥軒:
我很想要爆寫一波，但現在時間是要交專題計畫書當天的 6:19，下午還有統計專題發表，嗚嗚嗚。這一學期的資料庫管理讓我認識到，人的時間果然是有限的 XDDD