

# 資料庫管理 ( 114-1 )

## 期末專案完整報告

第 23 組

D10725001 江德偉、B10303113 楊珮筠、M11315039 徐唯剛

2025 年 12 月 10 日

GitHub 專案連結、展示影片連結

## 1 系統分析

災區、NGO、校園與社區單位經常臨時需要 1–3 小時的短期志工（例如活動人流引導、場地搬運、快閃陪讀等）。然而，傳統的招募方式（如透過通訊軟體群組）過程零散、資訊混亂，且難以有效控管名額、處理時間衝突或確保志工的技能符合需求。「Micro-Volunteering Pool」（極短期志工任務池）即是為了解決此窘境而設計的平台。

本系統旨在提供一個高效率的媒合管道，讓主辦單位（Organizer）能快速發布 1 至 3 小時的「微任務」，並讓志工（Volunteer）能根據自身的時間、地點與技能，自由報名參加。系統的核心功能在於處理快速的報名、退出、名額遞補，並透過技能配對機制提升任務的到位率與完成品質。

根據不同的功能及掌控權限，「Micro-Volunteering Pool」系統的用戶可以分為三種身分，分別是 Volunteer、Organizer 及 Admin。Volunteer（志工）：為系統的一般使用者。可依照自身需求，瀏覽平台上的任務，並選擇感興趣的進行報名。若任務額滿，可加入候補。亦可隨時退出已報名的任務（系統將自動遞補）。Volunteer 也能維護個人的技能組合與熟練度，以提高被主辦單位看見或配對的機會。Organizer（主辦）：為系統的業務經營者之一，負責發布及管理任務。可建立任務、設定任務所需的日期、連續時段（1-3 小時）、場地、名額上限，以及必要的技能需求。任務發布後，可管理報名與候補名單，並於任務結束後進行簽到與狀態回報。Admin（系統管理）：為系統的最高權限管理者，主要負責維護系統的基礎資料與全域設定，例如管理使用者帳號、主辦單位的資格、場地清單、以及全域的技能標籤庫。

### 1.1 系統功能

#### 1.1.1 關於任務的相關設定

系統上會提供可發布任務的時段，以 24 小時制的整數讓 Organizer 選擇。系統限定每一個時段訂為一小時。系統目前開放的最小任務時長以一小時為單位，一次預約至多三個時段，且這些時段必須連續。例如，若 Organizer 發起一場任務，預計日期在 2025/10/15，並選擇場地「校總區綜合體育館」。系統會顯示該場地可用的時段。假設 Organizer 決定任務時間為下午 2 點到 4 點（共 2 小時），則需選擇活動預約日期 2025/10/15、開始時間 14 與時間長度 2。系統將會記錄 14、15 兩個時段。

### **1.1.2 級 Volunteer 的功能**

在本系統中，Volunteer 可以執行以下功能：

1. 註冊與維護技能：使用者註冊後，可維護個人檔案，包含擁有的技能（如：基礎救護、活動主持）及其熟練度。
2. 篩選與查詢任務：使用者可依地點、日期、技能需求或容量狀態（是否額滿）來篩選平台上的任務。
3. 報名與候補任務：使用者可對未額滿的任務進行報名。若任務已額滿，可選擇加入候補隊列。
4. 退出任務：使用者如果臨時不想參加，可刪除已參與但未開始的任務報名資訊。系統將自動檢查候補隊列並依序遞補。
5. 查詢歷史任務：使用者可以查詢自身曾參與過的任務，包括已結束與正在進行中的。

### **1.1.3 級 Organizer 的功能**

在本系統中，Organizer 可以執行以下功能：

1. 建立任務：使用者能透過設定任務標題、描述、名額上限、場地、日期，以及 1 至 3 個連續時段來發起一場任務。同時可設定該任務所需的技能標籤。
2. 管理任務報名：可查詢任務的報名清單與候補隊列，並可查看報名者的技能配對分數。
3. 管理任務狀態：可手動編輯或取消尚未開始的任務，並於任務結束後標記狀態（如：完成、取消）及進行志工簽到。
4. 查詢統計：可查詢自己舉辦過的任務歷史紀錄，並觀看簡易統計（如：出席率、技能缺口）。

### **1.1.4 級 Admin 的功能**

在本系統中，Admin 可以執行以下功能：

1. 管理基礎資料：可對主辦單位（ORG）、場地（VENUE）、技能清單（SKILL）進行增刪改查的操作。
2. 管理使用者：可管理所有使用者的帳號狀態與權限（Volunteer, Organizer, Admin）。
3. 查詢全域紀錄：可查詢所有使用者的活動紀錄，以及所有任務的詳細資訊。

## 2 系統設計

### 2.1 ER Diagram

本系統「Micro-Volunteering Pool」的 ER Diagram 如圖 1 所示。其核心實體為 USER (使用者)，代表所有與「Micro-Volunteering Pool」平台互動的個體。所有使用者都必須註冊帳號才能使用平台功能。註冊時，系統會要求使用者提供基本資料如名稱、信箱與密碼，並在成功後指派一個唯一的 User\_id (使用者代號)。系統中的使用者身份 (Role) 具有彈性。一般使用者預設為 User (在此專案中又稱為 Volunteer)，但也可由後臺手動賦予 Admin (管理員) 或 Organizer (主辦) 權限，使其同時擁有多重身份。

圍繞著 USER，本 ERD 還定義了其他四個主要實體：TASK\_EVENT (任務)、SKILL (技能)、VENUE (場地) 和 ORG (主辦單位)。這些實體之間透過 USER\_SKILL、PARTICIPATION、MATCH\_SCORE、WAITLIST、OWNER\_ID、WHERE、HOST、TASK\_REQUIRED\_SKILL 等八個主要關係進行連結，共同構成了系統的資料結構。

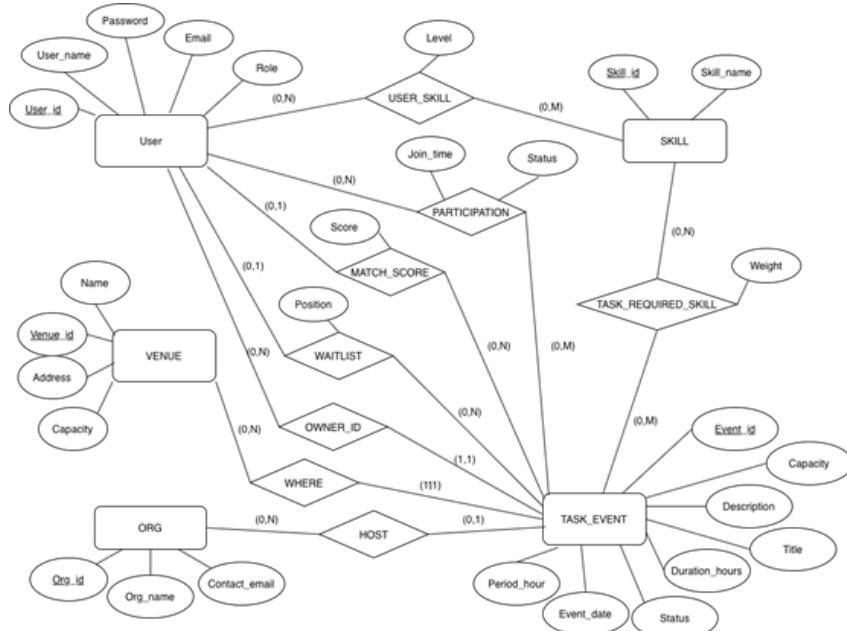


圖 1: 「Micro-Volunteering Pool」的 ER Diagram

TASK\_EVENT 代表一個志工任務，Organizer 可以發起多場任務，而 Volunteer 也可以報名參加多場任務。若 Organizer 想發起一場任務，系統會需要使用者提供該任務的資訊，包括任務的標題 (Title)、描述 (Description)、名額上限 (Capacity)，以及想招募的技能需求。其中名額上限與技能需求是發布任務的必填資訊。接著 Organizer 需要選擇任務舉辦的場地、確定的日期 (Event\_date) 與時段。系統限定任務時段為 1 至 3 個連續小時，此資訊會記錄在 TASK\_EVENT 實體的多值屬性 Period\_hour 中。系統也會記錄該任務的狀態 (Status)，例如「已規劃」、「進行中」或「已結束」。

系統會在 Organizer 發起任務時，讓其選擇可用的場地，其中場地用 VENUE 來表示。系統會呈現場地代號 (Venue\_id)、場地名稱 (Name)、地址 (Address) 及該場地可以容納的人數上限 (Capacity) 這些資訊。同時，一個任務必須歸屬於一個主辦單位，透過 ORG 來表示。系統會記錄主辦單位的代號 (Org\_id)、單位名稱 (Org\_name) 和聯絡信箱 (Contact\_email)。

Organizer 在發起任務時，可以指定該任務需要的特定技能，透過 SKILL 來表示。系統會記錄技能代號 (Skill\_id) 和技能名稱 (Skill\_name)。一個任務可以需求多項技能（透過 M:N 關係 TASK\_REQUIRED\_SKILL），而一個 Volunteer 也可以擁有多項技能（透過 M:N 關係 USER\_SKILL），並記錄其熟練度 (Level)。

此外，系統還包含多個關係來處理報名與媒合流程。PARTICIPATION 用於記錄志工的正式報名，包含報名時間 (Join\_time) 和狀態 (Status)。若任務額滿，志工的報名會進入 WAITLIST 關係，並記錄其候補順位 (Position)。MATCH\_SCORE 則是用於儲存系統計算出的志工與任務的技能配對分數 (Score)。

## 2.2 Relational Database Schema Diagram

### 2.2.1 基本設計

我們可以將圖 1 的 ER diagram 轉換成圖 2 的 Database Schema，一共由十二個關聯 (relation) 組成，分別是 USER、USER\_ROLE、ORG、VENUE、SKILL、TASK\_EVENT、TASK\_EVENT\_PERIOD、PARTICIPATION、USER\_SKILL、TASK\_REQUIRED\_SKILL、MATCH\_SCORE、WAITLIST。

USER、ORG、VENUE、以及 SKILL 這四個關聯是直接由同名的實體型態 (entity type) 轉換而來，其主鍵 (Primary key, PK) 分別是 User\_id、Org\_id、Venue\_id、和 Skill\_id。

USER\_ROLE 這個關聯則是由 USER 實體的 Role 延伸而來。因為其作為多值屬性 (multi-valued attribute)，所以必須獨立出來成為一個關聯。該關聯的主鍵是由 User\_id 和 Role 共同組成，其中 User\_id 同時也作為外部鍵 (Foreign key, FK) 參考到 USER 的主鍵 User\_id。

TASK\_EVENT 這個關聯的主鍵是 Event\_id，外部鍵則包括參考到 USER 主鍵的 Owner\_id、參考到 ORG 主鍵的 Org\_id，以及參考到 VENUE 主鍵的 Venue\_id。而由於 TASK\_EVENT 的 Period\_hour 屬性是多值屬性，所以我們將 TASK\_EVENT\_PERIOD 獨立出來成為一個關聯。TASK\_EVENT\_PERIOD 的主鍵由 Event\_id 和 Period\_hour 共同組成，同時 Event\_id 也作為外部鍵參考到 TASK\_EVENT 的主鍵。

PARTICIPATION 這個關聯是由多對多之 PARTICIPATION 關係型態產生而來。同理，USER\_SKILL、TASK\_REQUIRED\_SKILL、MATCH\_SCORE、和 WAITLIST 都是由 M:N 關係型態轉換而來。這些關聯的主鍵皆由兩個 (或多個) 外部鍵組合而成，分別參考到所連結的實體之主鍵 (例如 PARTICIPATION 的主鍵由 User\_id 和 Event\_id 組成)。

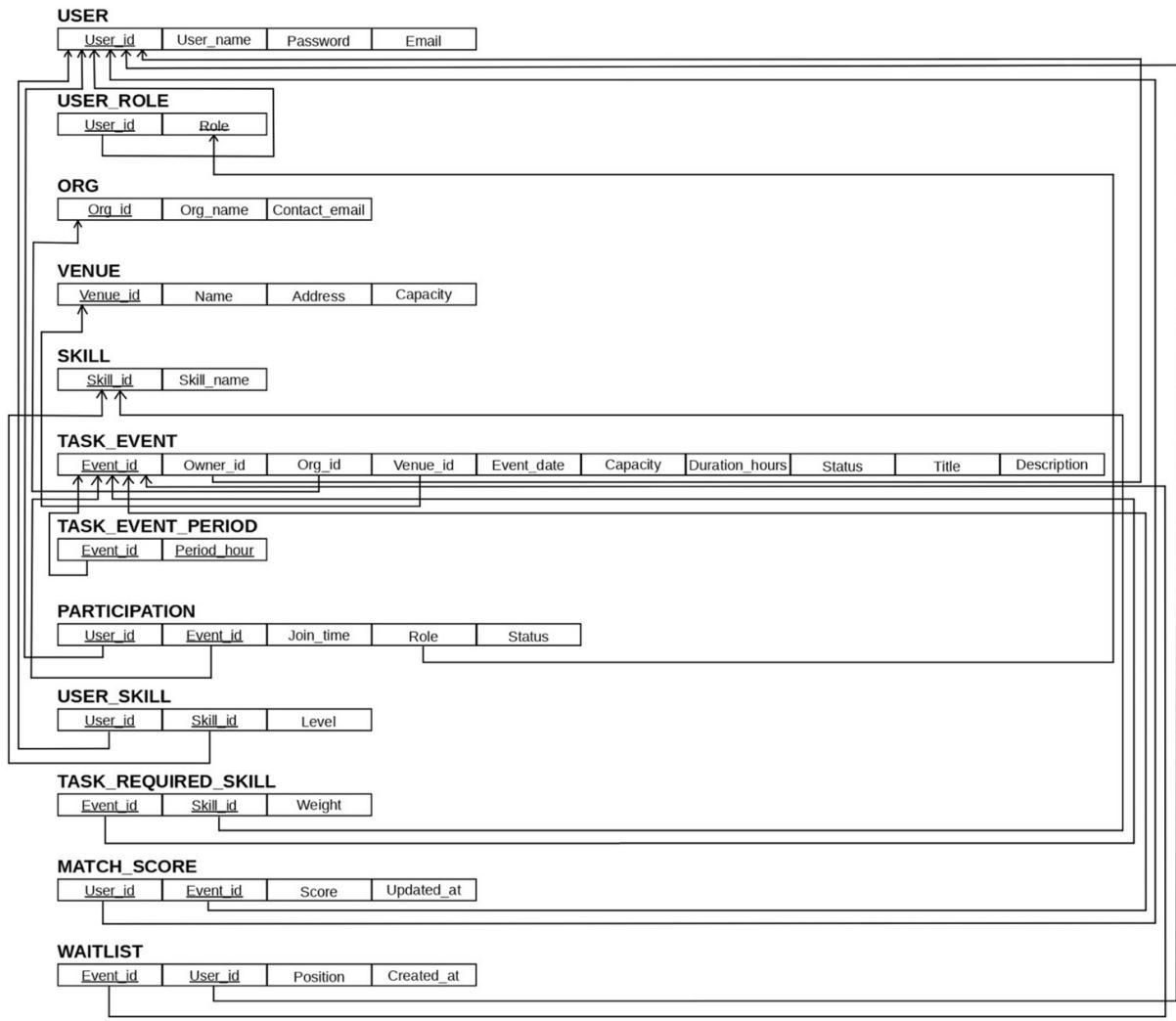


圖 2: 「Micro-Volunteering Pool」的 Relational Database Schema Diagram

### 2.3 Data Dictionary

「Micro-Volunteering Pool」的資料表共有圖 2 所示的 12 個，各個資料表的欄位相關資訊依序呈現  
在表 1 到表 12。

Column Name	Meaning	Data Type	Key	Constraint	Domain
User_id	使用者代號	bigint	PK	Not Null	
User_name	使用者名稱	Varchar (20)		Not Null, Unique	
Email	電子郵件	Varchar (50)			
Phone	電話號碼	int		Not Null	
Password	密碼	Varchar (60)		Not Null	

表 1: 資料表 USER 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
User_id	使用者代號	bigint	PK, FK: USER(User_id)	Not Null	
Role	使用者權限	Varchar (10)	PK	Not Null	
Referential triggers		On Delete	On Update		
User_id: USER(User_id)		Cascade	Cascade		

表 2: 資料表 USER\_ROLE 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
Org_id	主辦單位代號	bigint	PK	Not Null	
Org_name	單位名稱	Varchar (100)		Not Null	
Contact_email	聯絡信箱	varchar (100)		Not Null	

表 3: 資料表 ORG 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
Venue_id	場地代號	bigint	PK	Not Null	
Name	場地名稱	Varchar (10)		Not Null	
Address	地址	Varchar (200)		Not Null	
Capacity	可容納上線	Int		Not Null, >0	

表 4: 資料表 VENUE 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
Event_id	任務代號	bigint	PK	Not Null	
Owner_id	建立者代號	Bigint	FK: USER(User_id)	Not Null	
Org_id	主辦單位代號	Bigint	FK: ORG(Org_id)		
Venue_id	場地代號	Bigint	FK: VENUE(Venue_id)	Not Null	
Event_date	任務日期	Date		Not Null	
Capacity	名額上限	int		Not Null, >0	
Duration_hours	持續時數	int		Not Null	{1, 2, 3}
Status	任務狀態	Varchar (10)		Not Null	{Planned, Ongoing, Finished}
Title	標題	varchar (80)			
Description	描述	varchar (300)			
Referential triggers		On Delete	On Update		
Owner_id: USER(User_id)		Cascade	Cascade		
Org_id: ORG(Org_id)		Cascade	Cascade		
Venue_id: VENUE(Venue_id)		Cascade	Cascade		

表 5: 資料表 TASK\_EVENT 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
Event_id	任務代號	bigint	PK, FK: TASK_EVENT(Event_id)	Not Null	
Period_hour	任務舉辦時段	int	PK	Not Null	(0-23)
Referential triggers		On Delete	On Update		
Event_id: TASK_EVENT(Event_id)	Cascade	Cascade			

表 6: 資料表 TASK\_EVENT\_PERIOD 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
User_id	使用者代號	bigint	PK, FK: USER(user_id)	Not Null	
Event_id	任務代號	bigint	PK, FK: TASK_EVENT(Event_id)	Not Null	
Join_time	報名參與時間	datetime		Not Null	
Role	任務中角色	varchar (10)		Not Null	{Volunteer, Lead}
Status	參與狀態	varchar (10)		Not Null	{Active, Cancelled}
Referential triggers		On Delete	On Update		
User_id: USER(User_id)	Cascade	Cascade		Not Null	
Event_id: TASK_EVENT(Event_id)	Cascade	Cascade		Not Null	

表 7: 資料表 PARTICIPATION 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
Skill_id	技能代號	bigint	PK	Not Null	
Skill_name	技能名稱	varchar (60)		Not Null, Unique	

表 8: 資料表 SKILL 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
User_id	使用者代號	bigint	PK, FK: USER(User_id)	Not Null	
Skill_id	技能代號	bigint	PK, FK: SKILL(Skill_id)	Not Null	
Level	熟練度	int		Not Null	{1, 2, 3, 4, 5}
Referential triggers		On Delete	On Update		
User_id: USER(User_id)	Cascade	Cascade			
Skill_id: SKILL(Skill_id)	Cascade	Cascade			

表 9: 資料表 USER\_SKILL 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
Event_id	任務代號	bigint	PK, FK: TASK_EVENT(Event_id)		
Skill_id	技能代號	bigint	PK, FK: SKILL(Skill_id)		
Weight	需求權重	int		Not Null, Default 1, $\geq 1$	
Referential triggers		On Delete	On Update		
Event_id: TASK_EVENT(Event_id)		Cascade	Cascade		
Skill_id: SKILL(Skill_id)		Cascade	Cascade		

表 10: 資料表 TASK\_REQUIRED\_SKILL 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
User_id	使用者代號	bigint	PK,FK:USER(user_id)	Not Null	
Event_id	任務代號	bigint	PK, FK: TASK_EVENT(Event_id)	Not Null	
Score	配對分數	numeric (5,2)		Not Null	{1, 2, 3, 4, 5}
Updated_at	更新時間	timestamp		Not Null	
Referential triggers		On Delete	On Update		
User_id: USER(User_id)		Cascade	Cascade		
Event_id: TASK_EVENT(Event_id)		Cascade	Cascade		

表 11: 資料表 MATCH\_SCORE 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
User_id	使用者代號	bigint	PK, FK: USER(User_id)	Not Null	
Event_id	任務代號	bigint	PK, FK: TASK_EVENT(Event_id)	Not Null	
Position	候補順位	int		Not Null, $\geq 1$	
Created_at	建立時間	timestamp		Not Null	
Referential triggers		On Delete	On Update		
User_id: USER(User_id)		Cascade	Cascade		
Event_id: TASK_EVENT(Event_id)		Cascade	Cascade		

表 12: 資料表 WAITLIST 的欄位資訊

## 2.4 正規化分析

我們將依序從第一正規式 (1NF) 到第四正規式 (4NF)，來說明「Micro-Volunteering Pool」的關聯是如何滿足這些規則。

在 1NF 方面，我們的關聯中沒有任何一個屬性是 composite 或 multi-valued。在第 2.2 節中，我們將 multi-valued 屬性 Role (來自 USER) 及 Period\_hour (來自 TASK\_EVENT) 從原關聯中獨立出來，以獨立的關聯 (USER\_ROLE 及 TASK\_EVENT\_PERIOD) 描述，因此我們的 schema 滿足 1NF。

在 2NF 方面，關聯中的所有非鍵屬性 (non-prime attribute) 都完全功能相依 (fully functional dependency) 於任一候選鍵。以 PARTICIPATION 為例，其 Join\_time, Role, Status 屬性，皆完

全功能相依於複合主鍵 (`User_id`, `Event_id`)，沒有出現部分功能相依性，符合 2NF。

在 3NF 方面，一個關聯中的非鍵屬性都沒有遞移相依 (transitively dependency) 於主鍵。檢視我們設計的所有關聯，均符合 3NF。在比 3NF 更嚴謹的 BCNF 方面，要求關聯中的每一個功能相依的箭頭左方都要是超級鍵 (superkey)。我們的 schema 也符合 BCNF。

最後是 4NF，由於「Micro-Volunteering Pool」的所有關聯都不存在多值相依 (multi-valued dependency)，因此滿足 4NF 的條件。

## 3 系統實作

### 3.1 資料庫建置方式及資料來源說明

本系統使用 PostgreSQL 作為後端資料庫，所有資料表的結構定義於 `schema.sql`，並透過 `init_schema.py` 自動建立。執行初始化程式後，系統會在 `micro_volunteer` 資料庫中建立使用者 (USER)、角色 (USER\_ROLE)、場地 (VENUE)、主辦單位 (ORG)、志工技能 (SKILL)、活動 (TASK\_EVENT)、活動時段 (TASK\_EVENT\_PERIOD)、報名紀錄 (PARTICIPATION)、候補 (WAITLIST) 等多張資料表，作為整體系統的基礎結構。

系統初始資料並非以爬蟲取得，而是透過 `seed_disaster_data.py` 脚本加入數筆與實際情境相符的示範資料，例如：台灣災害救援中心、台南海灘淨灘、高雄臨時收容所支援等活動，以及多種志工技能 (急救、物資分類、海灘清潔等)。系統初始化後，共建立了約 10028 筆使用者資料、18 筆技能資料、221 筆示範活動、23 筆主辦單位資料、33 筆活動場地資料，並於後續操作中產生更多的報名與候補紀錄，做為測試媒合流程的基礎資料。

任務與技能需求同樣由腳本建立基本測試資料，後續的報名、取消、候補遞補等紀錄則會由使用者在 `client` 端操作時動態寫入資料庫。透過此資料初始化方式，系統能在開發與測試階段迅速建立一致的環境，並模擬真實的志工媒合流程。

### 3.2 重要功能及對應的 SQL 指令

第 1.1 節中我們已介紹本系統提供給志工 (User)、主辦方 (Organizer) 及管理員 (Admin) 的主要功能。在第 3.2.1 小節與第 3.2.2 小節中，我們將以特定情境為例，列出實作這些功能時所使用的 (一或數個) SQL 指令。除此之外，在第 3.2.3 小節中，我們也整理了系統層級的指令及其對應的 SQL 敘述。

#### 3.2.1 為 User 的功能

- 註冊志工帳號：若要實現此功能，假設情境為「使用者想註冊一個新的志工帳號，其使用者名稱 `user_name` 為『alex』，電子郵件 `email` 為『`alex@example.com`』，電話 `phone` 為『`0912345678`』，登入密碼 `password` 為『`pw123`』。」則對應的 SQL 指令如下。系統會在

"USER" 資料表新增一筆使用者資料，並由資料庫自動產生對應的 user\_id。

```
Insert Into "USER" (user_name, email, phone, password)
Values ('alex', 'alex@example.com', 0912345678, 'pw123')
Returning user_id;
```

Listing 1: 註冊志工帳號 SQL 指令

2. 查詢可加入的志工任務：若要實現此功能，假設情境為「某志工想查詢目前仍開放報名、尚有名額且尚未開始或進行中的任務。」對應 SQL 指令如下。系統會從 TASK\_EVENT 取出狀態為『Planned』或『Ongoing』的任務，並透過子查詢計算目前已報名人數，只回傳尚未額滿的任務資訊供使用者選擇。

```
Select *
From TASK_EVENT As te
Where te.status In ('Planned', 'Ongoing')
And (
    Select Count(*)
    From PARTICIPATION As p
    Where p.event_id = te.event_id
    And p.status = 'Active'
) < te.capacity;
```

Listing 2: 查詢可加入的志工任務 SQL 指令

3. 依日期與地點查詢淨灘任務：若要實現此功能，假設情境為「某志工想查詢日期為『2025-11-29』，地點在臺南地區，且任務標題中包含『淨灘』字樣的任務。」對應 SQL 指令如下。系統會從 TASK\_EVENT 與 VENUE 取得符合日期與地點條件的任務，並以關鍵字比對任務名稱。

```
Select te.*
From TASK_EVENT As te
Join VENUE As v On te.venue_id = v.venue_id
Where te.event_date = '2025-11-29'
And v.address Like '%台南%'
And te.title Like '%淨灘%';
```

Listing 3: 依日期與地點查詢淨灘任務 SQL 指令

4. 報名志工任務：若要實現此功能，假設情境為「志工代號 user\_id 『8』想報名任務代號 event\_id 『2』，並於 join\_time 『2025-11-28 10:00:00』提出申請。」若該任務尚有名額且尚未開始或進行中。

額，系統會在 PARTICIPATION 資料表新增一筆『Active』的參與紀錄，角色 role 設為『Volunteer』。對應的 SQL 指令如下。

```
Insert Into PARTICIPATION (user_id, event_id, join_time, role,
    status)
Values (8, 2, '2025-11-28 10:00:00', 'Volunteer', 'Active');
```

Listing 4: 報名志工任務 SQL 指令

5. 取消報名並觸發候補遞補：若要實現此功能，假設情境為「志工代號 user\_id 『8』想要取消任務代號 event\_id 『2』的報名。」系統會先將該志工在 PARTICIPATION 中的狀態標記為『Cancelled』，接著從 WAITLIST 中找出同一任務候補順位 position 最前面的一位志工，將其從候補名單移到 PARTICIPATION 成為正式志工。對應的 SQL 指令如下所示。

```
-- 將原志工標記為取消
Update PARTICIPATION
Set status = 'Cancelled'
Where event_id = 2 And user_id = 8;

-- 找出候補順位最前面的志工
With next_wait As (
    Select user_id
    From WAITLIST
    Where event_id = 2
    Order By position
    Limit 1
)

-- 將其加入正式參與名單
Insert Into PARTICIPATION (user_id, event_id, join_time, role,
    status)
Select user_id, 2, now(), 'Volunteer', 'Active'
From next_wait;

-- 自候補名單中刪除
Delete From WAITLIST
Where event_id = 2
```

```
|| And user_id In (Select user_id From next_wait);
```

Listing 5: 取消報名與候補遞補 SQL 指令

6. 查詢使用者曾經參與過的志工任務：若要實現此功能，假設情境為「志工代號 user\_id 『8』想要查看自己曾經參與過的所有志工任務。」則對應的 SQL 指令如下。系統會從 PARTICIPATION 與 TASK\_EVENT 取得該使用者所有『Active』或已取消的參與紀錄，並回傳對應的任務資訊。

```
|| Select te.*  
From PARTICIPATION As p  
Join TASK_EVENT As te On p.event_id = te.event_id  
Where p.user_id = 8  
And p.status In ('Active', 'Cancelled');
```

Listing 6: 查詢使用者曾參與志工任務 SQL 指令

### 3.2.2 給 Admin 的功能

1. 管理場地（增刪改查）

新增場地：若要實現此功能，假設情境為「某管理員想要新增一筆場地，其名稱 name 為『台南海灘活動區』，地址 address 為『Tainan Coastal Park』，可容納人數上限 capacity 為『50』。」則對應的 SQL 指令如下。系統會在 VENUE 資料表中新增一筆場地資訊。

```
|| Insert Into VENUE(name, address, capacity)  
Values ('台南海灘活動區', 'Tainan Coastal Park', 50);
```

Listing 7: 新增場地 SQL 指令

刪除場地：若要實現此功能，假設情境為「某管理員想刪除代號 venue\_id 為『3』的場地資訊。」則對應 SQL 指令如下。系統會在 VENUE 資料表中刪除該筆場地資料。

```
|| Delete From VENUE  
Where venue_id = 3;
```

Listing 8: 刪除場地 SQL 指令

更新場地：若要實現此功能，假設情境為「某管理員想更新場地代號 venue\_id 為『1』的場地資訊，將其可容納人數上限 capacity 改為『100』。」則對應 SQL 指令如下。

```
|| Update VENUE  
Set capacity = 100
```

```
|| Where venue_id = 1;
```

Listing 9: 更新場地 SQL 指令

查詢場地：若要實現此功能，假設情境為「某管理員想查詢地址包含『台南』字樣的所有場地資訊。」則對應的 SQL 指令如下。

```
|| Select *
  From VENUE
  Where address Like '%台南%';
```

Listing 10: 查詢場地 SQL 指令

2. 查詢使用者資訊：若要實現此功能，假設情境為「某管理員想要查詢使用者代號 user\_id 為『5』的使用者資訊。」則對應 SQL 指令如下。系統會回傳該使用者在 "USER" 資料表中的紀錄。

```
|| Select *
  From "USER"
  Where user_id = 5;
```

Listing 11: 查詢使用者資訊 SQL 指令

3. 查詢特定志工任務資訊：若要實現此功能，假設情境為「某管理員想查詢所有『淨灘』相關的志工任務。」則對應 SQL 指令如下。系統會從 TASK\_EVENT 資料表中找出任務標題包含『淨灘』的所有活動。

```
|| Select *
  From TASK_EVENT
  Where title Like '%淨灘%';
```

Listing 12: 查詢志工任務資訊 SQL 指令

### 3.2.3 系統級指令

1. 更新志工任務狀態：若要實現此功能，假設情境為「系統在每日批次作業中，將日期 event\_date 為『2025-11-29』且舉辦時段 period\_hour 為『14』的所有志工任務狀態 status 由『Ongoing』更新為『Finished』。」則對應的 SQL 指令如下。系統會利用 TASK\_EVENT 與 TASK\_EVENT\_PERIOD 的關聯，更新符合條件之任務的狀態欄位。

```
|| Update TASK_EVENT As te
  Set status = 'Finished'
```

```

From TASK_EVENT_PERIOD As tep
Where te.event_id = tep.event_id
And te.status = 'Ongoing'
And te.event_date = '2025-11-29'
And tep.period_hour = 14;

```

Listing 13: 更新志工任務狀態 SQL 指令

- 確認指定場地在日期與時段內是否已被預約：若要實現此功能，假設情境為「系統在主辦方建立新任務前，需要檢查場地代號 `venue_id` 為『1』的場地，在日期『2025-11-29』的時段『10–13』之間是否已有其他任務排程；若有重疊則回傳 1 ( `True` )，若沒有則回傳 0 ( `False` )。」則對應的 SQL 指令如下。這裡以時間區間重疊判斷是否已被預約。

```

Select
Case
When Exists (
    Select 1
    From TASK_EVENT As te
    Where te.venue_id = 1
        And te.event_date = '2025-11-29'
        And te.start_hour < 13 -- 與 [10,13) 有重疊
        And te.end_hour > 10
)
Then 1
Else 0
End;

```

Listing 14: 確認場地是否被預約 SQL 指令

- 重新計算指定任務的技能配對分數：若要實現此功能，假設情境為「系統在主辦方更新任務需求技能後，需重新計算任務代號 `event_id` 為『2』與所有志工之間的技能配對分數，並更新至 `MATCH_SCORE` 資料表。」我們假設配對分數為「志工技能等級 `level` 與任務需求權重 `weight` 的加權總和」。對應的 SQL 指令如下，使用 PostgreSQL 的 `On Conflict` 語法進行新增或更新。

```

Insert Into MATCH_SCORE (user_id, event_id, score, updated_at)
Select us.user_id,
       trs.event_id,
       Sum(us.level * trs.weight) As score,

```

```

    now() As updated_at
From USER_SKILL As us
    Join TASK_REQUIRED_SKILL As trs
        On us.skill_id = trs.skill_id
Where trs.event_id = 2
Group By us.user_id, trs.event_id
On Conflict (user_id, event_id)
Do Update
Set score      = Excluded.score,
updated_at   = Excluded.updated_at;

```

Listing 15: 重新計算技能配對分數 SQL 指令

### 3.3 交易管理

在本志工媒合系統中，多項功能會同時操作多個資料表，因此交易管理在整體架構中扮演極為重要的角色。我們採用 psycopg2 套件內建的交易機制，所有資料庫存取均透過 `get_conn()` 函式取得連線，並以 Python 的 `with` 區塊包裹實際執行的 SQL 指令。進入 `with` 區塊後，資料庫會自動開始一個交易，若區塊內的所有操作皆成功，該交易就會在離開區塊時自動 `commit()`；反之，若任一指令拋出例外，則會立即執行 `rollback()`，撤銷所有已執行的操作，確保資料的一致性與原子性。

在本系統的操作中，志工報名任務是一個典型需要交易管理的流程。報名動作包含檢查活動剩餘名額、寫入 `PARTICIPATION` 表、必要時新增至 `WAITLIST` 以及更新志工角色等程序。這些步驟必須同時成功才能維持資料的正確性，因此我們以單一交易包裝整個報名流程。一旦其中任何一步失敗，例如名額檢查與寫入紀錄結果不一致，系統便會進行 `rollback()`，避免產生「候補資料已新增但實際名額未更新」之類的不完整資料狀態。

另一次需要使用交易的案例是志工取消報名並觸發遞補邏輯。當志工取消參與時，系統會刪除其在 `PARTICIPATION` 中的紀錄，並同時從 `WAITLIST` 取出下一順位志工加入該任務。若遞補成功，系統會立即為候補志工新增一筆參與紀錄；若任何動作失敗，整個取消與遞補的過程皆會撤回，以確保名額與候補列表狀態一致。

此外，主辦方建立志工任務時，系統必須同時新增任務本體 (`TASK_EVENT`) 與該任務對應的多筆時段資訊 (`TASK_EVENT_PERIOD`)。這是一個需要跨表同步的操作，因此我們以交易確保在所有時段資料成功寫入之前，不會讓資料庫中僅出現「沒有時段的活動」這類不完整紀錄。同理，在新增任務所需技能 (`TASK_REQUIRED_SKILL`) 及重新計算志工配對分數 (`MATCH_SCORE`) 時，也會利用交易機制保持資料一致，讓所有技能需求或所有配對分數都在同一次交易中完成更新。

透過上述方式，本系統在志工報名、取消報名、候補遞補、任務建立與技能維護等多項流程中皆確保資料庫操作能維持正確性與一致性，使整體系統在多人同時使用的情況下仍能穩定運作。

### 3.4 併行控制

在本系統的設計中，主辦方在建立任務時所涉及的「場地與時段預約」是最容易發生併行衝突的環節。由於場地在同一時間只能提供一項活動使用，因此若有多位主辦方同時嘗試在相同場地的相同時段新增任務，就可能造成預約重疊的錯誤情形。這類問題往往源於查詢與新增之間的時間差，當使用者 A 查詢某場地某時段尚未被預約時，使用者 B 可能也在同一瞬間查到相同結果，若雙方幾乎同時按下「建立任務」，便會導致資料庫最後存入兩筆衝突的預約紀錄。

為避免此類併行問題，本系統在任務建立功能中加入了鎖定機制，採用 Python `threading` 模組的排他鎖（Lock）。在 `organizer.py` 中，`create_event()` 函式會在寫入資料庫之前先透過 `lock.acquire()` 取得鎖，確保在鎖被釋放之前，不會有其他建立任務的流程同時被執行。取得鎖之後，系統會再次檢查該場地在指定起訖時段內是否已被其他使用者預約，這能避免使用者看到的是舊查詢結果。若該時段尚未被占用，系統便會以交易方式寫入任務本體與對應的時段紀錄；若該場地在這段期間內已被其他任務預定，則系統會立刻釋放鎖並回傳錯誤訊息，提示使用者該時段已無法預約。

我們採用的鎖定時機是在使用者按下「確認建立任務」之後，而非在查詢可用時段時就加鎖。此作法能夠避免因使用者停留過久而造成不必要的等待，提升整體系統效能。最終，透過這套併行控制策略，即便在多人同時建立任務的情境下，仍能確保資料庫中不會出現場地或時段重複預約的問題，使整個系統在高度併行的環境中依然能維持正確性與可靠性。

## 4 分工資訊

- 徐唯剛：後端系統開發、交易管理與併行控制實作、SQL 核心指令。
- 楊珮筠：資料庫架構設計、正規化分析、業務邏輯流程的規劃與驗證。
- 江德偉：需求情境分析、簡報製作、書面報告撰寫與專案文件維護。

## 5 專案心得

徐唯剛：這次專案把整個志工媒合流程從資料庫 schema 到 TCP 伺服器、CLI 都走過一遍，實際體會到設計資料表時要兼顧業務邏輯（容量、候補、角色權限）與一致性（鎖、外鍵、檢查約束）。在並發報名/取消時遇到的超額問題，迫使我去加 `SELECT ... FOR UPDATE`，也學到「先鎖定資源再寫入」的重要性。雖然中途踩了不少坑（例如時間欄位改版、種子資料衝突），但也因此更熟悉了迭代 schema 與 seed 的流程，以及如何把 NoSQL（TinyDB）串進分析功能。

江德偉：這次專案讓我體驗到「先寫完簡單版本再逐步加功能」的價值，一開始只做基本註冊/搜尋/報名，後來再補 Organizer 管理、Admin 刪除、熱門關鍵字分析等需求。中途遇到的問題包括：TCP 伺服器回應過大導致 JSON 被截斷、角色大小寫導致權限判斷失效、以及多 client 併發時鎖不嚴導致超額。每解一個問題，就更清楚系統設計的邊界條件與防護點，也學到撰寫 README、初始化腳本、種子資料的重要性，讓別人接手或部署時能快速上手。

楊珮筠：這次的專案讓我體驗了從無到有的系統分析過程，最大的挑戰在於如何將現實生活中抽象的志工媒合概念，轉化為精確的資料庫 Schema，也讓我深刻體會到理論知識與實際開發的落差。在設計 Schema 時，不僅要考慮儲存資料，更要思考如何支援後端的交易管理，我們也為此討論、修改了很多次。看到原本紙上的設計圖，最終修正並落實為能實際使用的系統，讓我非常有成就感。感謝組員的合作及配合，從這門課跟專案學習到了很多。