資料庫管理（113-1）
期末專案完整報告

Group 31
B10801011 張鈞傑、B11801004 卓庭榛
2024 年 12 月 11 日

影片連結：https://youtu.be/CSAKQKuRKSQ?si=u39i3elzZSw_CwQz
GitHub 連結：https://github.com/cccalan1108/Bello_system

---

1 系統分析

想認識新朋友？不想被限制於現在的交友圈？還是週末無聊想找一個陌生人喝一杯？又或者想透過語言交換結交在台灣的外國朋友呢？這些需求歡迎使用「Bello」。

Bello 是卡通人物小小兵語言當中「你好」，在義大利文當中亦有「美好、帥氣、漂亮」之意，期望大家皆可以向外踏出認識新朋友，有個 Bello Life！

雖然現今的手機、社群平台相當發達，但在 Facebook、Instagram、Threads 等平台上的活動久而久之是否有一種「空虛」的感覺呢？在社群平台上的人都是素不相識的人，人們的相識初次可能透過演算法無意間地瀏覽到他 / 她的貼文、Reels 影片，覺得有趣、長得好看所以追蹤，但從此便僅止於追蹤，如果有一個機會可以出來聊天、真的認識一下或許更好？

然而，想進一步認識的話又該如何邀約呢？如若想認識自己生活交際圈外的新朋友，想吃個飯、喝點酒、聊聊天，在大眾的留言區留言太過暴露，但直接傳訊息給別人好像也有些奇怪，倘若有一個平台可以正當地發起交友、見面、聚會，或許便不會尷尬。在 Bello，使用者將可透過舉辦、參加活動的方式來和平台上的陌生朋友相見。

此外，隨著國際化的趨勢，國人學習外語的情況大幅增加，然而，學習語言不應僅止於讀、寫，必須透過實際的對話來增進聽、說的能力，且人們自發性學習新的語言其實也是對學習語言的國家文化和人民有相當程度的興趣而學，因此透過實際與該母語人士的對話來練習是更好不過的選擇，在學習的同時也結交新的朋友。現今在臉書上偶有中英、中日、中韓語言交換的聚會，大家相約在咖啡廳玩遊戲、聊天來交換語言，通常是由某個機構、語言補習班、粉絲專頁社團舉辦的，所以其實很多人也不知道有這類資訊。或許透過具體的平台，中文母語者可參與欲學習語言之交換語言聚會、在台灣想中文、認識台灣人的外國人亦可選擇自己母語的聚會參加 (若無則通常選擇英語)。

本系統提供兩種身份，分別為 User 和 Admin。Admin 主要為本系統的相關研發團隊及業務人員，可管理所有使用者的資訊、聚會；User 則為一般使用者。使用者在註冊時可輸入與自己有關的資訊，像是姓名、暱稱、性別、生日、星座、MBTI、血型、社群帳號、自我介紹、興趣、想尋找的聚會，使用者如若想「發起」新的一次性聚會作為主召人，可選擇要發起「用餐、咖啡、飲酒、語言交換」類型的聚會、聚會目的 (輸入原因即可，像是想跟新朋友吃飯、想跟日本人用日語對話…等)、聚會日期、聚會時間、城市、地點、上限人數；而使用者若想「參與」聚會，可以在「聚會」的頁面瀏覽，若看見想加入的聚會，只需輸入前述相關資訊即可報名。

1.1 系統功能

1.1.1 關於交友平台的相關設定

使用者在註冊時可自行設定登入用的帳號、密碼, 並輸入基本資訊 (真實姓名、使用者名稱、性別、國籍、居住城市、手機、Email、生日、國籍), 其中使用者名稱是用於在平台上顯示的名稱, 可使用真實姓名、英文名字或暱稱。

在個人介紹資訊方面, 每位使用者皆可自行選擇是否輸入星座、血型、MBTI、宗教、大學、婚姻狀態、自我介紹、興趣、想尋找的聚會類型、是否公開社群平台 SNS 帳號資訊等資訊, 而若選擇公開自己提啊社交平台帳號資訊供平台上的使用者瀏覽的話, 也可選擇分別要公開哪些社交平台的帳號資訊。

每一個使用者皆可成為聚會舉辦者, 在系統中可以舉辦聚會, 每場聚會皆可選擇其聚會內容 / 目的 (午餐、咖啡/下午茶、晚餐、喝酒、語言交換聊天), 且每場聚會皆可指定、選擇可使用的語言 (中文, 台語, 客語, 原住民語, 英文, 日文, 韓文, 法文, 德文, 西班牙文, 俄文, 阿拉伯文, 泰文, 越南文, 印尼文), 舉辦者在新增活動時需同時輸入舉會城市、地點 (地址或店名)、日期、時間、人數上限, 聚會地點由主辦者自行決定。而每一位使用者也都可以自行選擇要加入哪些感到興趣的聚會, 在平台上點選加入即可。

平台提供聊天室的服務給使用者, 避免大家在不熟悉、不認識的情況下就先交換私人聯絡方式以免造成困擾。使用者彼此之間可透過平台的聊天室以私人訊息聊天室的方式聯絡, 而同一場聚會的參與者也會有該場聚會專屬的聊天室供參與者聊天。

1.1.2 給 User 的功能

在本系統中, User 可以執行以下功能:

1. 單純聊天 (使用者可透過一對一私人聊天室、同屬一場聚會的共同聊天室進行對話)
2. 舉辦聚會 (午餐 / 咖啡、下午茶 / 晚餐 / 語言交換 / 喝酒)
3. 參加聚會會 (午餐 / 咖啡、下午茶 / 晚餐 / 語言交換 / 喝酒)
4. 取消聚會會 (午餐 / 咖啡、下午茶 / 晚餐 / 語言交換 / 喝酒)
5. 查詢目前平台中可參加的聚會
6. 查詢使用者本人先前舉辦過的聚會
7. 查詢使用者本人先前參加過的聚會

1.1.2 給 Admin 的功能

在本系統中, Admin 可以執行以下功能:

1. 管理現有聚會:Admin可針對現有且不符規定之聚會做刪改。
2. 管理聊天室:可將具惡意或不符規定之用戶自聊天室移除, 以及禁言用戶等。
3. 查詢使用者資訊:可查詢使用者在註冊時填入之個人資訊、用戶曾參與過之聚　會以及聊過天之使用者。
4. 查詢聊天資訊:可查詢使用者間之聊天內容及相關資訊。

2 系統設計

2.1 ER Diagram

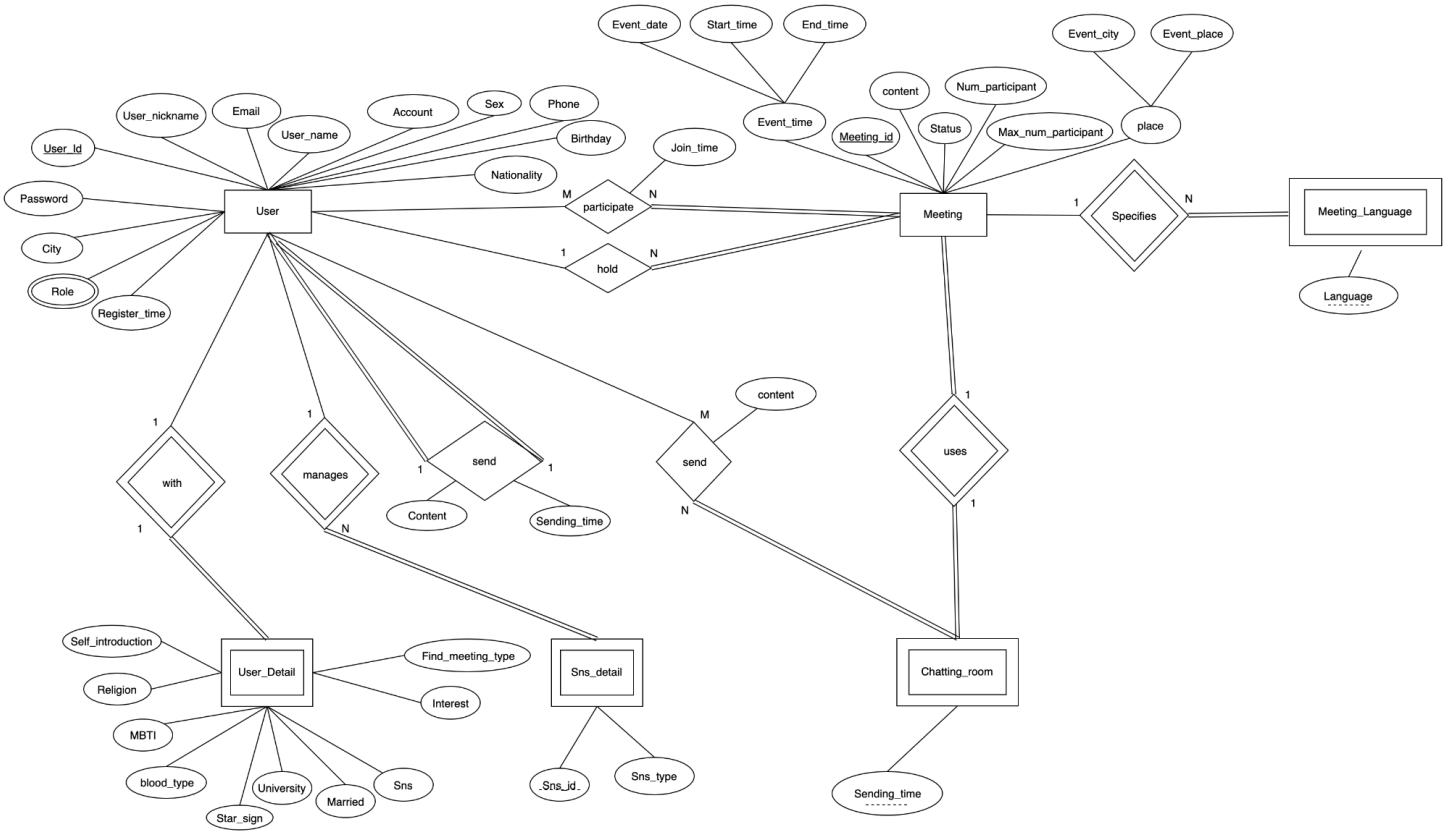

圖 1:「Bello」的 ER diagram

圖 1 為「Bello」之 ER diagram, 共有 6 個 Entity, 其中 2 個為 Strong entity, 分別為 USER、MEETING; 另外 4 個為 Weak entity, 分別為 USER_DETAIL、SNS_DETAIL、CHATTING_ROOM、MEETING_LANGUAGE。

ER diagram 中共有 8 個 Relationship, 分別為 With、Manages、Send、Send、Uses、Specifies、Hold、Participate。每名 User 除了註冊時填入之基本資料外, 還能自由選擇填一個關於興趣嗜好之 USER_DETAIL, User 經過後臺手動設定，可以將特定使用者也設定為 Admin，此時這個使用者就會同時有 User 和 Admin 兩個身份。此外，User 可能擁有並管理社交平台帳號，但未必所有 User 皆有在用社交平台，而 User 可以選擇已私聊的方式傳送訊息給其他 User 或是透過 Meeting 之聊天室傳送訊息，訊息皆會紀錄傳送時間及內容。

User 還可以決定要舉辦 Meeting 或是參與 Meeting，其選擇參與 Meeting 的時間會被記錄下來，且一位 User 可以參與多個聚會，一個聚會也可以有多個 User，但一場聚會的舉辦者僅能有一位 User。每個 Meeting 會有其語言限制，因為一場 Meeting 可能有多種語言，故為一對多之關係，且一個 Meeting 必有一個聊天室。Meeting 除了相關資訊外，還會有人數上限，以及參與人數之紀錄。

2.2 Relational Database Schema Diagram

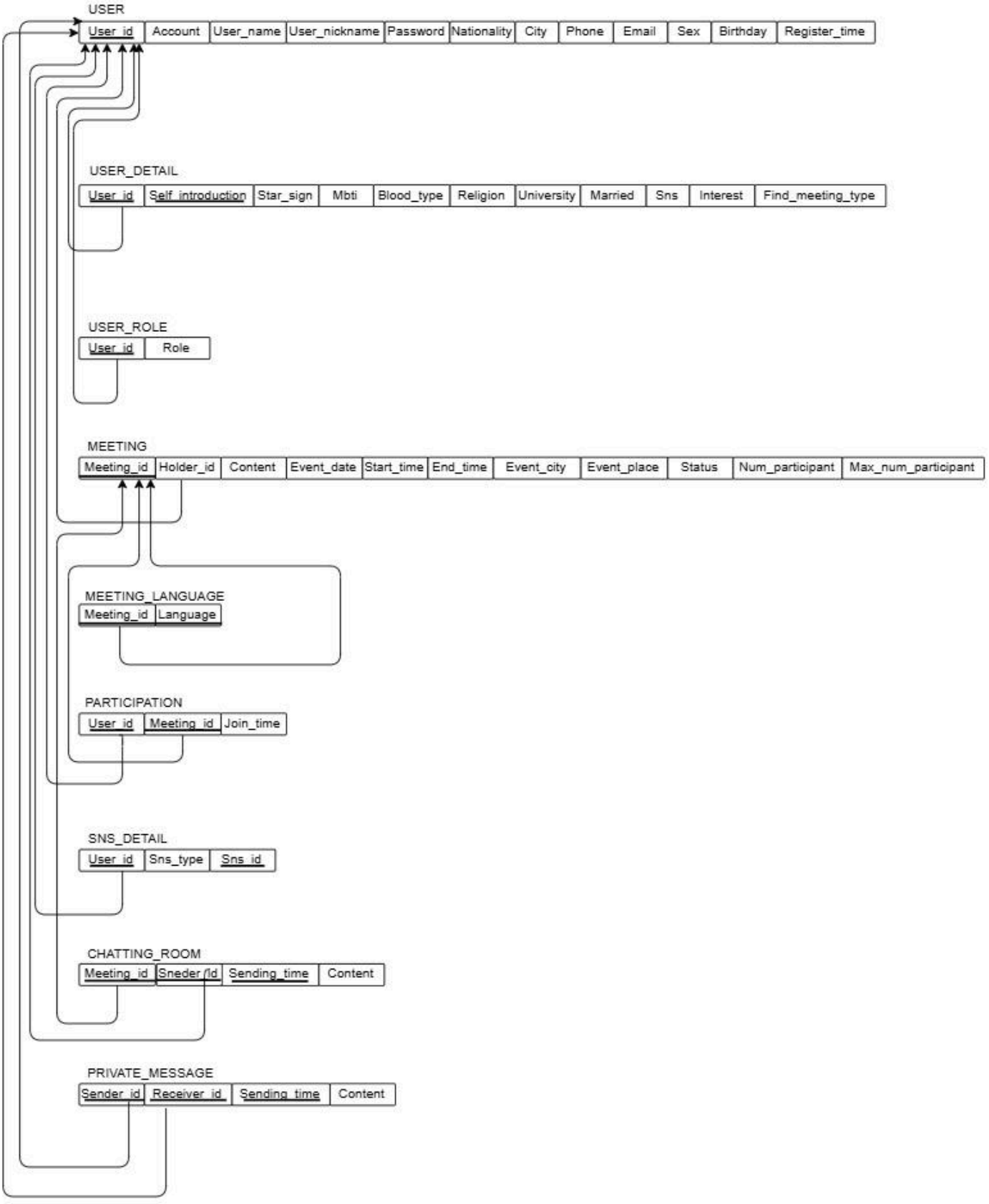

圖 2:「Bello」的 Relational Database Schema Diagram

圖 2 為「Bello」的 Relational Database Schema Diagram, 在 USER 中以每位使用者的編號 User_id 作為主鍵; 在 USER_DETAIL、USER_ROLE 則以 User_id 作為外部鍵, 參考到 USER 主鍵的 User_id。在 MEETING 中以每場聚會的編號 Meeting_id 作為主鍵, 以 User_id 作為外部鍵, 參考到 USER 主鍵的 User_id。

在 MEETING_LANGUAGE 當中, 由於每場聚會可以使用多種語言、具多值屬性之性質, 因此將其獨立出來做為一個 Relationship, 其中 Meeting_id 同時作為主鍵及外部鍵參考到 MEETING 主鍵的 Meeting_id。而每場聚會的報名資訊皆會被記錄下來, 在 PARTICIPATION 當中以 User_id 和 Meeting_id 同時作為主鍵, User_id 參考到 USER 主鍵的 User_id、Meeting_id 則參考到 MEETING 主鍵的 Meeting_id。

在聚會的共同聊天室 CHATTING ROOM 中, 儲存每場聚會的所有使用者在聊天室發送的對話紀錄, 以 Meeting_id、Sender_id、Sending_time 同時作為主鍵, Meeting_id 參考到 MEETING 主鍵的 Meeting_id、Sender_id 參考到 USER 主鍵的 User_id。同時, 私人聊天室 PRIVATE_MESSAGE 中, 則以 Sender_id、Receiver_id、Sending_time 同時作為主鍵, 兩者皆作為外部鍵參考到 USER 主鍵的 User_id。

2.3 Data Dictionary

「Bello」資料表共有 9 個, 個資料表的相關資訊依序呈現在下方表 1 至表 9。

Column Name	Meaning	Data Type	Key	Constraint	Domain
User_id	使用者編號	bigint	PK	Not Null, Unique	
Account	使用者帳號	varchar (15)		Not Null, Unique	
User_name	使用者真實姓名	varchar (20)		Not Null	
User_nickname	使用者名稱	varchar (20)		Not Null	
Password	使用者密碼	varchar (15)		Not Null	
Nationality	國籍	varchar (20)		Not Null	
City	居住城市	varchar (20)			
Phone	手機	varchar (20)		Not Null, Unique	
Email	信箱	varchar (50)		Not Null, Unique	
Sex	性別	varchar (10)		Not Null	
Bitthday	生日	date		Not Null	
Register_time	註冊時間	datetime		Not Null	

表 1:資料表 USER 欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
User_id	使用者編號	bigint	PK, FK : USER(User_id)	Not Null, Unique	
Self_introduction	自我介紹(必填)	varchar (200)	PK		
Star_sign	星座	varchar (20)			{摩羯, 水瓶, 雙魚, 牡羊, 金牛,

					雙子, 巨蟹, 獅子, 處女, 天秤, 天蠍, 射手}
Mbti	MBTI	varchar (5)			{ISTP, ISFP, ESTP, ESFP, ISTJ, ISFJ, ESTJ, ESFJ, INTP, INTJ, ENTP, ENTJ, INFJ, INFP, ENFJ, ENFP}
Blood_type	血型	varchar (5)			{A, B, AB, O}
Religion	宗教	varchar (15)			{無, 佛教, 道教, 基督教, 天主教, 伊斯蘭教, 印度教, 其他}
University	大學	varchar (50)			
Married	婚姻狀態	varchar (5)			{未婚, 已婚, 喪偶}
Sns	社交軟體資訊是否開放讓其他使用者知道	varchar (5)			{YES, NO}
Interest	興趣	varchar (200)			
Find_meeting_type	想尋找的聚會	varchar (50)			
Referential triggers		On Delete	On Update		
User_id : USER (User_id)		Cascade	Cascade		

表 2:資料表 USER_DETAIL 欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
User_id	使用者編號	bigint	PK, FK : USER(User_id)	Not Null, Unique	
Role	使用者權限	varchar (10)	PK	Not Null	{User, Admin}
Referential triggers		On Delete	On Update		
User_id : USER (User_id)		Cascade	Cascade		

表 3:資料表 USER_ROLE 欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
Meeting_id	聚會編號	bigint	PK	Not Null, Unique	
Holder_id	舉辦者編號	bigint	FK : USER(User_id)	Not Null	
Content	聚會內容	varchar (50)		Not Null	{午餐, 咖啡/下午茶, 晚餐, 喝酒, 語言交換}
Event_date	舉會日期	date		Not Null	
Start_time	開始時間	time		Not Null	
End_time	結束時間	time		Not Null	
Event_city	城市	varchar (20)		Not Null	

Event_place	聚會地點	varchar (50)	Not Null	
Status	聚會狀態	varchar (20)	Not Null	{Ongoing, Finished, Canceled}
Num_participant	報名人數	int	Not Null	
Max_num_participant	人數上限	int		

Referential triggers	On Delete	On Update
Holder_id : USER (User_id)	Cascade	Cascade

表 4:資料表 MEETING 欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
Meeting_id	聚會編號	bigint	PK, FK : MEETING (Meeting_id)	Not Null	
Language	語言	vaarchar (40)	PK	Not Null	{中文, 台語, 客語, 原住民語, 英文, 日文, 韓文, 法文, 德文, 西班牙文, 俄文, 阿拉伯文, 泰文, 越南文, 印尼文 }

Referential triggers	On Delete	On Update
Meeting_id : MEETING (Meeting_id)	Cascade	Cascade

表 5:資料表 MEETING_LANGUAGE 欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
User_id	使用者編號	bigint	PK, FK : USER (User_id)	Not Null	
Meeting_id	聚會編號	bigint	PK, FK : MEETING (Meeting_id)	Not Null	
Join_time	報名時間	datetime		Not Null	

Referential triggers	On Delete	On Update
User_id : USER (User_id)	Cascade	Cascade
Meeting_id : MEETING (Meeting_id)	Cascade	Cascade

表 6:資料表 PARTICIPATION 欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
User_id	使用者編號	bigint	PK, FK : USER (User_id)	Not Null	
Sns_type	有使用且公開給其他使用者瀏覽的社群軟體	varchar (10)		Not Null	{Facebook, Instagram, Threads, X, Tiktok, 小紅書, WhatsApp, LINE, WeChat, KakaoTalk}

Sns_id	有使用且公開給其他使用者瀏覽的社群軟體之帳號 ID	varchar (20)	PK	Not Null

Referential triggers	On Delete	On Update
User_id : USER (User_id)	Cascade	Cascade

表 7:資料表 SNS_DETAIL 欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
Meeting_id	聚會編號	bigint	PK, FK : MEETING (Meeting_id)	Not Null	
Sender_id	訊息發送者編號	bigint	PK, FK : USER (User_id)	Not Null	
Sending_time	訊息發送時間	datetime	PK		
Content	訊息內容	varchar (200)			

Referential triggers	On Delete	On Update
Meeting_id : MEETING (Meeting_id)	Cascade	Cascade
Sender_id : USER (User_id)	Cascade	Cascade
Receiver_id : USER (User_id)	Cascade	Cascade

表 8:資料表 CHATTING_ROOM 欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
Sender_id	訊息發送者編號	bigint	PK, FK : USER(User_id)	Not Null	
Receiver_id	訊息接收者編號	bigint	PK, FK : USER(User_id)	Not Null	
Sending_time	訊息發送時間	datetime	PK		
Content	訊息內容	varchar (200)			

Referential triggers	On Delete	On Update
Sender_id : USER (User_id)	Cascade	Cascade
Receiver_id : USER (User_id)	Cascade	Cascade

表 9:資料表 PRIVATE_MESSAGE 欄位資訊

2.4 正規化分析

在正規化 (Normalization) 分析當中，下方將依序由第一正規化 (1NF) 到第四正規化 (4NF) 依照定義來分析「Bello」平台資料庫是否滿足各正規化之規則。

第一正規化 1NF 的要求每個欄位都必須是單一值，且資料表中的每一列都應該是唯一的，每張資料表的每個欄位都為原子值，例如各表中的 User_id、Role、Meeting_id 等都代表單一值。又如 MEETING_LANGUAGE 表中，Language 欄位已分成單一的語言值，如中文、英文等，而不是多個語言組合成的集合。因此，所有資料表均符合 1NF。

第二正規化 2NF 需符合 1NF，並且消除部分依賴，即所有非主鍵屬性必須完全依賴於主鍵。以 PARTICIPATION 表為例，由 User_id 和 Meeting_id 組成主鍵，且每個非主鍵屬性都完全依賴於組合主鍵，符合 2NF。又以 SNS_DETAIL 表為例，由 User_id 和 Sns_type 組成主鍵，Sns_id 完全依賴於這個組合鍵，符合 2NF。在此標準下，所有資料表皆滿足 2NF 的條件。

第三正規化 3NF 需符合 2NF，並且消除傳遞依賴，即所有非主鍵屬性不能依賴於其他非主鍵屬性。以 USER_DETAIL 中的 Sns 為例，並非每位使用者都願意公開 SNS，而若有公開的話才需要相關的 SNS 資訊，為避免傳遞依賴，我們將 SNS 的相關訊息從 USER_DETAIL 中移出，其他資料表中皆無傳遞依賴，符合 3NF。

BCNF 則是需要符合 3NF，並且所有決定因素（即決定其他屬性的屬性組合）都是超鍵 (Superkey)。例如 USER_ROLE 表中，User_id 和 Role 組成的主鍵決定了使用者的角色，無其他依賴，符合 BCNF。又如 CHATTING_ROOM 表中，主鍵由 Meeting_id 和 Sender_id 決定，其所有非主鍵屬性如 Sending_time、Content依賴於此主鍵，符合 BCNF。所有表的決定因素均為超鍵，因此符合 BCNF。

第四正規化的要求是在符合 BCNF 的基礎上，消除多值依賴，即一個主鍵不應同時對應多個相互無關的多值屬性。MEETING_LANGUAGE 表中，Meeting_id 對應多個語言值 Language，但此關聯已經分離成獨立表，符合 4NF。SNS_DETAIL 表中，每個 User_id 對應不同的社交平台帳號和 ID，這些已分成單獨記錄，消除多值依賴；在 SNS_DETAIL 表中，每位使用者皆可以公開多個社群軟體的帳號資訊，無多值依賴。其餘資料表亦無發生此多值依賴情況，故我們資料表亦符合 4NF。

3 系統實作

3.1 資料庫建置方式及資料來源說明

本專案中最重要的資料表便為 USER 資料表，用於儲存平台使用者的基本資料，包括帳號（Account）、姓名（User_name）、暱稱（User_nickname）、國籍（Nationality）等欄位。我們透過匯入模擬的 CSV 資料檔案（users.csv），共產生 10,000 筆使用者資料。此資料表設置多個資料驗證條件，像是密碼（Password）的長度限制、電話號碼和 Email 的唯一性，確保資料的正確性與完整性，也避免相同的手機號碼和 Email 重複註冊。

```
CREATE DATABASE bello;
\c bello;
-- Create tables for Bello platform
SET client_encoding = 'UTF8';
-- Create USER table
CREATE TABLE "USER" (
    User_id BIGINT PRIMARY KEY,
    Account VARCHAR(15) NOT NULL UNIQUE,
    User_name VARCHAR(20) NOT NULL,
    User_nickname VARCHAR(20) NOT NULL,
    Password VARCHAR(15) NOT NULL,
    Nationality VARCHAR(20) NOT NULL,
    City VARCHAR(20),
    Phone VARCHAR(20) NOT NULL,
    Email VARCHAR(50) NOT NULL UNIQUE,
    Sex VARCHAR(10) NOT NULL,
    Birthday DATE NOT NULL,
    Register_time TIMESTAMP NOT NULL
);
\copy "USER" FROM 'users.csv' WITH (FORMAT csv);
```

圖 3.1.1、USER 資料表建置

USER_DETAIL 資料表是根據使用者的相關個人資料則是透過主頁中「個人資料」來修改個人的個人介紹、星座、MBTI、宗教、血型等相關資訊來生成，資料表中的 User_id 欄位通過 FOREIGN KEY 與 USER 資料表的

User_id 進行聯結，並設置 ON DELETE CASCADE 與 ON UPDATE CASCADE 條件，確保在 USER 表資料變更或刪除時，USER DETAIL 資料能夠同步更新或刪除。

```sql
-- Create USER_DETAIL table
CREATE TABLE USER_DETAIL (
    User_id BIGINT REFERENCES "USER"(User_id) ON DELETE CASCADE ON UPDATE CASCADE,
    Star_sign VARCHAR(20) CHECK (Star_sign IN ('摩羯', '水瓶', '雙魚', '牡羊', '金牛', '雙子',
    '巨蟹', '獅子', '處女', '天秤', '天蠍', '射手')),
    Mbti VARCHAR(5) CHECK (Mbti IN ('ISTP', 'ISFP', 'ESTP', 'ESFP', 'ISTJ', 'ISFJ', 'ESTJ', 'ESFJ',
    'INTP', 'INTJ', 'ENTP', 'ENTJ', 'INFJ', 'INFP', 'ENFJ', 'ENFP')),
    Blood_type VARCHAR(5) CHECK (Blood_type IN ('A', 'B', 'AB', 'O')),
    Religion VARCHAR(15) CHECK (Religion IN ('無', '佛教', '道教', '基督教', '天主教', '伊斯蘭教', '印度教', '其他')),
    University VARCHAR(50),
    Married VARCHAR(5) CHECK (Married IN ('未婚', '已婚', '喪偶')),
    Sns VARCHAR(5) CHECK (Sns IN ('YES', 'NO')),
    Self_introduction VARCHAR(200),
    Interest VARCHAR(200),
    Find_meeting_type VARCHAR(50),
    PRIMARY KEY (User_id, Self_introduction)
);
```

圖 3.1.2、USER_DETAIL 資料表建置

USER_ROLE 資料表用於記錄每位使用者的角色資訊，確保平台能夠區分一般使用者與管理員，並提供相應的權限控制功能。使用 User_id 和 Role 的組合作為複合主鍵，確保每位使用者的角色記錄唯一，支持一位使用者擁有多種角色的設計，並通過 FOREIGN KEY 與 USER 資料表中的 User_id 欄位進行關聯，用於標識角色所屬的使用者。

```sql
-- Create USER_ROLE table
CREATE TABLE USER_ROLE (
    User_id BIGINT REFERENCES "USER"(User_id) ON DELETE CASCADE ON UPDATE CASCADE,
    Role VARCHAR(10) CHECK (Role IN ('User', 'Admin')),
    PRIMARY KEY (User_id, Role)
);
```

圖 3.1.3、USER_ROLE 資料表建置

MEETING 資料表用於儲存平台上的聚會資訊，包括聚會的舉辦者(Holder_id)、活動內容(Content)、活動日期(Event_date)、活動地點(Event_place)、狀態(Status)等欄位。設計上透過外鍵與 USER 資料表建立連結，確保每筆聚會資料對應到合法的使用者。聚會的最大參與人數(Max_num_participant)及當前參與人數(Num_participant)設為可控欄位，提供彈性資料操作。

```sql
-- Create MEETING table
CREATE TABLE MEETING (
    Meeting_id BIGINT PRIMARY KEY,
    Holder_id BIGINT REFERENCES "USER"(User_id) ON DELETE CASCADE ON UPDATE CASCADE,
    Content VARCHAR(50) NOT NULL CHECK (Content IN ('午餐', '咖啡/下午茶', '晚餐', '喝酒', '語言交換')),
    Event_date DATE NOT NULL,
    Start_time TIME NOT NULL,
    End_time TIME NOT NULL,
    Event_city VARCHAR(20) NOT NULL,
    Event_place VARCHAR(50) NOT NULL,
    Status VARCHAR(20) NOT NULL CHECK (Status IN ('Ongoing', 'Finished', 'Canceled')),
    Num_participant INTEGER NOT NULL,
    Max_num_participant INTEGER
);
```

圖 3.1.4、MEETING 資料表建置

MEETING_LANGUAGE 資料表作為 MEETING 資料表的附屬資料，設計用於記錄每場聚會的語言需求，方便參與者根據語言偏好選擇適合的活動，提升活動匹配度。Meeting_id 作為外鍵，與 MEETING 資料表中的 Meeting_id 欄位進行關聯，標識語言需求所屬的聚會。Language：此欄位用於記錄聚會所使用的語言，透過 CHECK 條件限制有效的語言選項。

```sql
-- Create MEETING_LANGUAGE table
CREATE TABLE MEETING_LANGUAGE (
    Meeting_id BIGINT REFERENCES MEETING(Meeting_id) ON DELETE CASCADE ON UPDATE CASCADE,
    Language VARCHAR(40) CHECK (Language IN ('中文', '台語', '客語', '原住民語', '英文', '日文', '韓文',
    '法文', '德文', '西班牙文', '俄文', '阿拉伯文', '泰文', '越南文', '印尼文')),
    PRIMARY KEY (Meeting_id, Language)
);
```

圖 3.1.5、MEETING_LANGUAGE 資料表建置

PARTICIPATION 資料表記錄了每位使用者參與聚會的歷史資料，通過外鍵分別與 USER 及 MEETING 資料表進行連結，確保資料的參照完整性。此資料表中加入參與時間欄位，便於後續進行行為分析。

```sql
-- Create PARTICIPATION table
CREATE TABLE PARTICIPATION (
    User_id BIGINT REFERENCES "USER"(User_id) ON DELETE CASCADE ON UPDATE CASCADE,
    Meeting_id BIGINT REFERENCES MEETING(Meeting_id) ON DELETE CASCADE ON UPDATE CASCADE,
    Join_time TIMESTAMP NOT NULL,
    PRIMARY KEY (User_id, Meeting_id)
);
```

圖 3.1.6、PARTICIPATION 資料表建置

SNS_DETAIL 資料表作為 USER 資料表的附屬資料，用於記錄使用者的社群帳號資訊，方便系統管理與使用者之間的交流匹配，為平台提供多元的聯絡與互動方式。使用 User_id、Sns_type 和 Sns_id 的組合作為複合主鍵，確保每位使用者的社群資料唯一。

```sql
-- Create SNS_DETAIL table
CREATE TABLE SNS_DETAIL (
    User_id BIGINT REFERENCES "USER"(User_id) ON DELETE CASCADE ON UPDATE CASCADE,
    Sns_type VARCHAR(10) CHECK (Sns_type IN ('Facebook', 'Instagram', 'Threads', 'X',
    'Tiktok', '小紅書', 'WhatsApp', 'LINE', 'WeChat', 'KakaoTalk')),
    Sns_id VARCHAR(20) NOT NULL,
    PRIMARY KEY (User_id, Sns_type, Sns_id)
);
```

圖 3.1.7、SNS_DETAIL 資料表建置

CHATTING_ROOM 資料表為 MEETING 資料表和 USER 資料表的關聯資料，設計用於記錄參與者在某場聚會聊天室中的交流訊息，提供系統即時聊天功能，確保每則訊息均能被正確儲存、快速檢索查詢。其中 Meeting_id 和 Sender_id 欄位設置 ON DELETE CASCADE 和 ON UPDATE CASCADE 條件，當聚會或使用者被刪除或更新時，自動同步刪除或更新相關的聊天記錄。而 Content 欄位的字元限制確保訊息內容在合理範圍內，避免資料表因儲存大段文字而影響性能。

```sql
-- Create CHATTING_ROOM table
CREATE TABLE CHATTING_ROOM (
    Meeting_id BIGINT REFERENCES MEETING(Meeting_id) ON DELETE CASCADE ON UPDATE CASCADE,
    Sender_id BIGINT REFERENCES "USER"(User_id) ON DELETE CASCADE ON UPDATE CASCADE,
    Sending_time TIMESTAMP,
    Content VARCHAR(200),
    PRIMARY KEY (Meeting_id, Sender_id, Sending_time)
);
```

圖 3.1.8、CHATTING_ROOM 資料表建置

PRIVATE_MESSAGE 資料表設計用於記錄平台用戶之間的私密消息交流，提供用戶之間一對一即時通訊的數據支持，確保每條私訊均能被正確存儲與查詢。使用 Sender_id、Receiver_id 和 Sending_time 的組合作為複合主鍵，確保每條私訊在發送者和接收者之間具有唯一性。使用 Sender_id、Receiver_id 和 Sending_time 的組合作為複合主鍵，確保每條私訊在發送者和接收者之間具有唯一性。Content 欄位的字元限制確保訊息內容在合理範圍內，避免儲存大段文字影響性能。

```sql
-- Create PRIVATE_MESSAGE table
CREATE TABLE PRIVATE_MESSAGE (
    Sender_id BIGINT REFERENCES "USER"(User_id) ON DELETE CASCADE ON UPDATE CASCADE,
    Receiver_id BIGINT REFERENCES "USER"(User_id) ON DELETE CASCADE ON UPDATE CASCADE,
    Sending_time TIMESTAMP,
    Content VARCHAR(200),
    PRIMARY KEY (Sender_id, Receiver_id, Sending_time)
);
```

圖 3.1.9、PRIVATE_MESSAGE 資料表建置

3.2 重要功能及對應的 SQL 指令

在 1.1 節的部分我們有介紹關於本系統提供給 User 和 Admin 的功能，在 3.2.3、3.2.4 小節中將介紹後端系統在實作上完成這些功能使用之 SQL 指令和相關 Python 程式、及程式間的運行邏輯。

圖 3.2.0、Bello 交友暨語言交換聚會系統前端之功能頁面

3.2.1 註冊功能

Bello 系統提供使用者網頁註冊功能，檢查資料完整性與帳號唯一性後，將基本資料存入資料庫並分配角色。若提供管理員驗證碼，角色設為 Admin，否則為 User，角色資訊儲存於 USER_ROLE 表。資料檢查與插入方面使用者需提供基本資訊，系統驗證完整性與帳號是否重複，若不符即返回錯誤。驗證後，系統將資料存入 USER 表、分配唯一 User_id、並記錄註冊時間。效能上透過索引加快帳號與電子郵件檢查速度，採用連線池減少資源競爭，提升高併發穩定性。此外，後端僅返回必要狀態訊息，降低網路流量與系統負載。

```python
def check_account_exists(self, account):
    query = 'SELECT COUNT(*) FROM "USER" WHERE Account = %s'
    result = self.execute_query(query, (account,))
    return result[0][0] > 0
```

圖 3.2.1 - 1、確保註冊資料不重複之驗證 SQL 指令

```
# 插入用戶資料
insert_query = """
    INSERT INTO "USER" (
        User_id, Account, Password, User_name, User_nickname,
        Nationality, City, Phone, Email, Sex, Birthday, Register_time
    )
    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
    RETURNING User_id
"""
```

圖 3.2.1 - 2、註冊時插入資料表之 SQL 指令

```
def set_user_role(self, user_id, role):
    try:
        role_query = """
            INSERT INTO "user_role" (User_id, Role)
            VALUES (%s, %s)
        """
        self.execute_query(role_query, (user_id, role))
        self.conn.commit()
        return True
    except Exception as e:
        self.conn.rollback()
        print(f"Error setting user role: {e}")
        return False
```

圖 3.2.1 - 3、設定 User Role 之 SQL 指令

3.2.2 登入功能

允許已註冊的使用者透過輸入帳號與密碼進入系統，接收後查詢資料庫，進行驗證以確保安全性。系統在 USER 資料表中為 Account 和 Password 欄位建立索引，加速資料定位與配對、降低查詢延遲。此外，透過單一查詢完成帳密驗證及基本資料的提取，減少多次資料庫連線所耗費的資源。

```
def verify_login(self, account, password):
    query = """
            SELECT u.User_id, u.User_name, u.User_nickname, u.Email, r.Role
            FROM "USER" u
            JOIN "user_role" r ON u.User_id = r.User_id
            WHERE u.Account = %s AND u.Password = %s
            """
    result = self.execute_query(query, (account, password))
```

圖 3.2.2、登入 SQL 指令

3.2.3 給 User 的功能

1. 新增聚會：使用者皆可發起新的聚會，將資料分別插入 MEETING、MEETING_LANGUAGE 和 PARTICIPATION 三個資料表。系統會透過 SQL 查詢自動計算最大的 Meeting_id 並遞增，確保每筆聚會資料具唯一識別碼。後端處理中，系統檢查請求資料是否符合要求，並確保活動時間合理。語言資料透過迴圈逐項插入 MEETING_LANGUAGE 表。效能方面，系統採用批次操作將語言資料一次插入資料表，避免重複連線資料庫。此外，活動 ID 查詢使用 MAX(Meeting_id) 優化計算，並將多表插入操作整合於單一資料庫交易中，有效減少資源消耗。

```
with self.conn.cursor() as cur:
    # 先獲取最大的 Meeting_id
    get_max_id_query = """
        SELECT COALESCE(MAX(Meeting_id), 0) + 1
        FROM MEETING
    """
    cur.execute(get_max_id_query)
    new_meeting_id = cur.fetchone()[0]

    # 插入聚會基本信息
    insert_meeting_query = """
        INSERT INTO MEETING (
            Meeting_id,
            Holder_id,
            Content,
            Event_date,
            Start_time,
            End_time,
            Event_city,
            Event_place,
            Status,
            Num_participant,
            Max_num_participant
        ) VALUES (
            %s, %s, %s, %s, %s, %s, %s, %s, 'Ongoing', 1, %s
        ) RETURNING Meeting_id
    """
```

圖 3.2.3 - 1 - 1 、新增聚會功能 SQL 指令

```
# 插入聚會語言
for language in meeting_data["languages"]:
    insert_language_query = """
        INSERT INTO MEETING_LANGUAGE (Meeting_id, Language)
        VALUES (%s, %s)
    """
    cur.execute(insert_language_query, (meeting_id, language))

# 創建者自動加入聚會
insert_participation_query = """
    INSERT INTO PARTICIPATION (Meeting_id, User_id, Join_time)
    VALUES (%s, %s, NOW())
"""
```

圖 3.2.3 - 1 - 2 、插入語言 SQL 指令

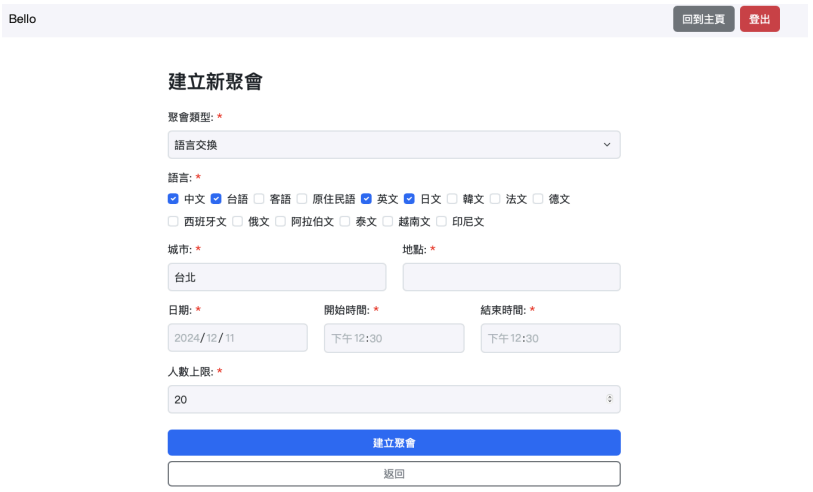

圖 3.2.3 - 1 - 3 、新增聚會之前端頁面

2. 加入聚會:當使用者請求加入聚會時，系統首先檢查該聚會是否尚有空位，透過查詢 MEETING 資料表的當前參與人數與最大允許人數，確保聚會未達上限。若聚會可加入，系統將使用者的參與資訊插入 PARTICIPATION 表，並更新 MEETING 表中的參與人數。

```
# 檢查聚會是否已滿
check_query = """
    SELECT Num_participant, Max_num_participant
    FROM MEETING
    WHERE Meeting_id = %s
    FOR UPDATE
"""
```

圖 3.2.3 - 2 - 1 、加入聚會前檢查人數是否已滿之 SQL 指令

```
# 插入參與記錄
insert_query = """
    INSERT INTO PARTICIPATION (User_id, Meeting_id, Join_time)
    VALUES (%s, %s, NOW())
"""
self.execute_query(insert_query, (user_id, meeting_id))

# 更新聚會人數
update_query = """
    UPDATE MEETING
    SET Num_participant = Num_participant + 1
    WHERE Meeting_id = %s
"""
```

圖 3.2.3 - 2 - 2 、加入聚會之 SQL 指令

3. 聊天室列表：快速查看與其他聊天對象的最近交流情況，使用 SQL 的 WITH 子查詢，通過檢索 private_message 表，將當前用戶的所有聊天對象提取出來，並透過查詢 sending_time 對每個聊天對象檢索最新的消息時間，且按時間排列、提取最新記錄，並將聊天對象的名稱從 USER 資料表中取回，並將結果按照最後消息的發送時間排序，且 NULLS LAST 確保沒有聊天記錄的對象列於末尾。

```
"""獲取用戶的聊天列表"""
try:
    cursor = self.conn.cursor()
    query = """
        WITH chat_partners AS (
            SELECT DISTINCT
                CASE
                    WHEN pm.sender_id = %s THEN pm.receiver_id
                    ELSE pm.sender_id
                END as partner_id
            FROM private_message pm
            WHERE pm.sender_id = %s OR pm.receiver_id = %s
        )
        SELECT
            cp.partner_id,
            u.user_name,
            (
                SELECT sending_time
                FROM private_message pm2
                WHERE (
                    (pm2.sender_id = %s AND pm2.receiver_id = cp.partner_id)
                    OR
                    (pm2.sender_id = cp.partner_id AND pm2.receiver_id = %s)
                )
                ORDER BY sending_time DESC
                LIMIT 1
            ) as last_message_time
        FROM chat_partners cp
        JOIN "USER" u ON u.user_id = cp.partner_id
        ORDER BY last_message_time DESC NULLS LAST
```

圖 3.2.3 - 3、聊天室列表查詢之 SQL 指令

4. 用戶之間的聊天記錄查詢：系統接受兩個用戶的 ID（user1_id 和 user2_id），並執行查詢，檢索他們的聊天記錄。SQL 查詢包含了發送者 ID、接收者 ID、訊息內容、發送時間以及發送者的姓名。為了提高查詢效率，訊息按照發送時間降序排列，並限制返回的訊息數量為 20 條。

```
def get_user_chat_history(self, user1_id, user2_id, limit=20):
    try:
        query = """
        SELECT
            pm.sender_id,
            pm.receiver_id,
            pm.sending_time,
            pm.content,
            u.user_name as sender_name
        FROM private_message pm
        INNER JOIN "USER" u ON pm.sender_id = u.user_id
        WHERE (pm.sender_id = %s AND pm.receiver_id = %s)
        OR (pm.sender_id = %s AND pm.receiver_id = %s)
        ORDER BY pm.sending_time DESC
        LIMIT %s
        """
```

圖 3.2.3 - 4、用戶之間的聊天記錄查詢 SQL 指令

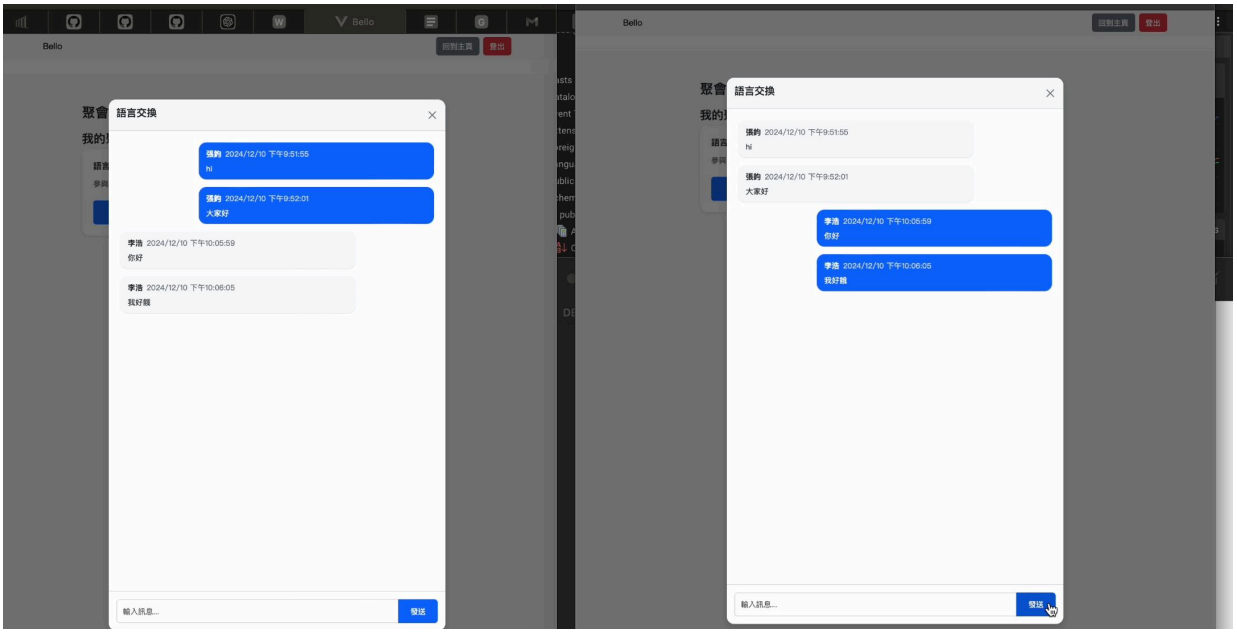

圖 3.2.3 - 5、用戶私人、聚會對話之前端介面

圖 3.2.3 - 6、聚會列表之前端介面

5. 查詢在線用戶：查詢並返回當前用戶之外的可用用戶清單，從資料表 "USER" 中選取用戶的 User_id 和暱稱，並排除管理員用戶和當前使用者，以便獲取一份其他普通在線用戶的名單。

```
def get_available_users(self, current_user_id):
    query = """
        SELECT u.User_id, u.User_nickname as nickname
        FROM "USER" u
        LEFT JOIN user_role ur ON u.User_id = ur.User_id AND ur.Role = 'Admin'
        WHERE u.User_id != %s AND ur.User_id IS NULL
        ORDER BY u.User_nickname
    """
    result = self.execute_query(query, (current_user_id,))
    if not result:
        return []

    users = []
    for row in result:
        users.append({"user_id": row[0], "nickname": row[1]})
    return users
```

圖 3.2.3 - 7、查詢在線用戶 SQL 指令

6. 聊天系統中的用戶搜尋功能:使用者可以在聊天系統中搜尋想找的使用者,在搜尋欄輸入一部分用戶名稱或暱稱後,系統即刻返回符合條件的用戶清單,供用戶進行選擇。

```
def search_chat_users(self, query, current_user, limit=10):
    try:
        sql = """
            SELECT User_id, User_name, User_nickname
            FROM "USER"
            WHERE (User_id::text LIKE %s
                   OR User_nickname LIKE %s
                   OR User_name LIKE %s)
            AND User_id != %s
            LIMIT %s
        """
```

圖 3.2.3 - 8、聊天系統中的用戶搜尋功能 SQL 指令

3.2.4 給 Admin 的功能

1. 獲取所有聚會:此功能提供管理員檢索系統中所有聚會、參與者的能力,包含聚會基本信息及參與者清單,以便管理員有效管理聚會。檢索系統內所有聚會記錄中,查詢 MEETING 表,提取聚會所有資訊,並按狀態 (Ongoing、Finished、Cancelled)、日期排列。並對每個聚會,執行獨立查詢,從 PARTICIPATION 表、USER 表中提取該聚會的參與者列表。

```
def get_all_meetings_admin(self):
    try:
        query = """
            SELECT
                m.Meeting_id,
                m.Content,
                m.Event_date,
                m.Event_place,
                m.Status,
                m.max_num_participant,
                m.Holder_id,
                u.User_name as holder_name,
                m.Num_participant
            FROM MEETING m
            LEFT JOIN "USER" u ON m.Holder_id = u.User_id
            ORDER BY
                CASE m.Status
                    WHEN 'Ongoing' THEN 1
                    WHEN 'Finished' THEN 2
                    WHEN 'Cancelled' THEN 3
                END,
                m.Event_date DESC
```

圖 3.2.4 - 1、Admin 獲取所有聚會和參與者資訊 SQL 指令

2. 取消聚會、將聚會標記為完成:管理員可以直接取消聚會,將指定的進行中聚會 (Status 為 Ongoing) 標記為已取消 (Canceled)、已完成 (Finished)。

```
def admin_cancel_meeting(self, meeting_id):
    try:
        query = """
                UPDATE MEETING
                SET Status = 'Canceled'
                WHERE Meeting_id = %s
                AND Status = 'Ongoing'
                """
```

圖 3.2.4 - 2、Admin 取消聚會 SQL 指令

3.2.5 系統級指令

1. 聚會聊天室紀錄查詢：系統接收 meeting_id進行精準查詢，透過資料庫中的 chatting_room 表，匹配相應的聊天記錄，並使用 INNER JOIN 操作獲取發送者的名稱，如此每條訊息都能與其發送者信息對應，實現資料的清晰關聯性。

```
def get_meeting_chat_history(self, meeting_id):
    try:
        query = """
        SELECT
            mm.meeting_id,
            mm.sender_id,
            u.User_name,
            mm.content,
            mm.sending_time
        FROM chatting_room mm
        INNER JOIN "USER" u ON mm.sender_id = u.User_id
        WHERE mm.meeting_id = %s
        ORDER BY mm.sending_time ASC
        """
```

圖 3.2.5 - 1、聚會聊天室紀錄查詢 SQL 指令

2. 獲取用戶 SNS 資訊：從資料表 SNS_DETAIL 中查詢 sns_type（SNS 的類型）和 sns_id（SNS 對應的帳號 ID）。使用條件 WHERE User_id = %s，僅查詢與傳入的 user_id 對應的資料，使用條件 WHERE User_id = %s，僅查詢與傳入的 user_id 對應的資料。

```
def get_sns_details(self, user_id):
    try:
        sql = """
            SELECT sns_type, sns_id
            FROM SNS_DETAIL
            WHERE User_id = %s
        """
        result = self.execute_query(sql, (user_id,))
        return [{"sns_type": row[0], "sns_id": row[1]} for row in result]
    except Exception as e:
        print(f"Error in get_sns_details: {str(e)}")
        return []
```

圖 3.2.5 - 2、獲取用戶 SNS 資訊 SQL 指令

3.3 SQL 指令效能優化與索引建立分析

3.3.1 在 USER 資料表中加入帳號與密碼檢查

為了提升使用者登入功能的效能，我們針對系統頻繁驗證用戶提交帳號與密碼的需求進行了優化。在登入過程中，系統主要依賴 USER 資料表中的 Account 和 Password 欄位進行查詢。然而，未經優化的查詢可能導致資料庫回應延遲，進而影響整體使用者體驗。為了解決這一問題，我們在 USER 資料表中針對 Account 和 Password 欄位設置了一個複合索引。此舉旨在提升查詢效能，減少資料庫的檢索時間，使登入操作更加快速。建立該索引的語法如下：

對比了此項改變前後的查詢速度與穩定性。在未設置索引的情況下，進行了 5 次模擬查詢測試，其平均執行時間為 1.345 秒。而在設置索引後，同樣進行了 5 次測試，結果顯示平均執行時間顯著縮短至 0.542 秒。測試

結果表明，通過為關鍵欄位設置索引，我們成功地將查詢時間顯著縮短，並且查詢結果的穩定性也得到了提升。這項優化措施不僅提高了系統效能，還為使用者提供了更加流暢的登入體驗。

3.3.2 資料庫分區

為了提升資料庫效能，我們針對 MEETING 表的資料按狀態 (Status) 分區管理和查詢，根據聚會狀態將資料分為 **Ongoing (**進行中**)**、**Finished (**已完成**)**、**Canceled (**已取消**)**。這種分區方式的目的是減少查詢過程中無關資料的掃描範圍，從而提高查詢速度，特別是在進行狀態過濾的操作時 (例如只查詢「進行中」的會議)，分區的實現主要依賴於 PostgreSQL 的分區功能。我們針對分區前後的查詢效能進行測試，在未進行分區的情況下，查詢所有進行中會議的平均執行時間為 1.204 秒，而插入新資料的平均執行時間為 0.453 秒。相比之下，在分區後的環境中，查詢所有進行中會議的平均執行時間明顯降低，而插入新資料的平均執行時間則基本保持不變，為 0.457 秒。

這些結果表示分區對於特定狀態的查詢效能提升達顯著，尤其是在資料量龐大的情況，分區的優勢更加明顯。同時，分區對插入操作的效能幾乎沒有影響，確保了系統在提高查詢速度時仍保持穩定的插入效能，進一步驗證了分區作為優化大資料量資料庫的一種有效策略，在資料檢索和系統效能上帶來優化。

3.4 交易管理與併行控制

在 User 要上傳自己的自我介紹時，會呼叫 update_user_detail，而其中我們 try catch 來處理例外，系統會先檢查此 User 是否已有關於其自我介紹的相關紀錄，若有則抓出，無則於資料庫中建一筆關於此 User 自我介紹之資料，而後將 User 更新之資料傳至資料庫。而這過程中，若有任何問題如資料庫未連上、傳入的參數不對、又或是系統臨時中斷連線，則會進入 except 執行 rollback，將所有已執行的部分回歸原狀，並回傳錯誤訊息，沒問題才會 submit 以確保資料上傳不出錯。

```python
def update_user_detail(self, field, value, user_id):
    try:
        cursor = self.conn.cursor()

        # 檢查用戶是否已有詳細資料記錄
        check_query = """
            SELECT Self_introduction FROM USER_DETAIL WHERE User_id = %s
        """
        cursor.execute(check_query, (user_id,))
        result = cursor.fetchone()

        if result:
            # 如果有記錄,更新現有記錄
            update_query = f"""
                UPDATE USER_DETAIL
                SET {field} = %s
                WHERE User_id = %s AND Self_introduction = %s
            """
            cursor.execute(update_query, (value, user_id, result[0]))
        else:
            # 如果沒有記錄,插入新記錄
            # 注意:這裡需要同時插入 Self_introduction,因為它是主鍵的一部分
            insert_query = """
                INSERT INTO USER_DETAIL (User_id, Self_introduction, {})
                VALUES (%s, '', %s)
            """.format(field)
            cursor.execute(insert_query, (user_id, value))

        self.conn.commit()
        return True

    except Exception as e:
        print(f"Error updating user detail: {str(e)}")
        self.conn.rollback()
        return False
```

圖 3.4、更新使用者資訊 SQL 指令

4 分工資訊

- 張鈞傑:基本上做所有工作、前端開發及前後端資料庫設計和串接、書面報告
- 卓庭榛:基本上做所有工作、前端開發及後端資料庫及各函數、影片錄製、書面報告

5 專案心得

- 張鈞傑:原本三人組別變成兩人實在很累，但透過這次的專案讓我們能夠自己撰寫一整套完整的資料庫系統，並且從開發者、使用者多方角度去設計跟修改的經驗很有意義，而第一次接觸前後端的串接，雖然很累也很艱辛，但我們兩個真的也學到了很多新的技術，實作果真是學習上最有助益的方法，看到成果還是覺得這是一個很值得的專案。
- 卓庭榛:在架設 Bello 系統的過程中，我認為最大的挑戰是串接前後端，畢竟先前都沒有這種開發系統的經驗，而在老師沒特別細講的情況下只能自己上網自學，雖然很痛苦，但卻讓我感到很有收穫，學到很多新技術，也做到了以前認為做不到的事，之後若再遇到類似的開發系統專案時，我相信能更得心應手！