資料庫管理（113-1）
期末專案完整報告範例

第 0 組
D11725004 胡捷翔、B10902080 胡家榆
2024 年 11 月 20 日
GitHub 專案連結、展示影片連結 [1]

1 系統分析

作業寫不出來卻沒人可以一起討論？期中快到了卻沒人可以一起衝刺？如果你有這些困擾不用緊張，趕緊上「I'm in」尋找你的最佳戰友！

「I'm in」是一個提供給某大學學生發起讀書會的平台，主要目的是幫助該校學生在特定日期和時段內，透過租借教室、召開一次性讀書會，以解決修課時，若遇到課程問題需要及時媒合同學討論卻苦無方法的窘境。所謂的「一次性讀書會」是指一種特定形式的學習活動，專爲大學學生提供的平台上的主要活動類型。在這種讀書會活動中，每次讀書會都是獨立的，組成該讀書會的成員可以是同一群人或不同；每次討論的課程內容也可以相同或不同。但是無論組成成員或討論內容是否相同，只要每次討論就需要重新發起一個新的讀書會，每個讀書會的編號也都不同。換言之，本系統中的「讀書會」是一個活動而非組織。

根據不同的功能及掌控權限，「I'm in」系統的用戶可以分爲兩種身分，分別是 User 及 Admin。若作爲一般使用者，身分別定義爲 User，可依照自身需求，選擇發起讀書會的活動，或是參與平台上由其他人發起的讀書會。若選擇發起讀書會，則可以透過介面輸入讀書會的內容、讀書會預計招攬的人數上限，以及想討論的是哪堂課的課程等資訊，接著選擇想租借的教室，並確定預約日期與時段。如果使用者只想參與其他人發起的讀書會，則可瀏覽平台上現有的讀書會，並選擇感興趣的進行加入。而 Admin 則是「I'm in」系統的業務經營者，主要負責管理課程及租借教室的資訊，並且可查詢所有使用者的活動紀錄，包括該使用者曾經發起及參與的讀書會有哪些、該讀書會的詳細資訊爲何等等，並據此做分析以優化平台營運。另外在第 1.1.4 小節中，我們也列出幾個不是針對特定使用者或管理員的功能，而是系統本身運行及維護所需的功能。

[1] 展示影片的需求請以 NTU COOL 作業區的「期末專案任務敘述」爲準。

1.1 系統功能

1.1.1 關於讀書會的相關設定

系統上會提供教室開放時間的時段，以 24 小時制的整數讓使用者選擇，可選範圍則是系統設定的教室開放時間[2]，目前設定爲 8 到 21。系統限定每一個時段訂爲一小時，亦即若使用者在系統上預約 13 點，則表示可從下午一點使用到下午兩點，其他依此類推。系統目前開放的最小預約時段以一小時爲單位，一次預約至多三個時段，且這些時段必須連續。例如若某使用者發起一場編號爲 A 的讀書會，接著他想爲這場讀書會的活動預約日期在 2023/10/12，並選擇教室名稱「共同 301」，此時系統會顯示這間教室目前可供預約時段，假設共有 13、14、15、16、19 五個可預約時段，則此時讀書會 A 可以預約 14、15、16 三個時段，也可以預約 13、14、15 三個時段，但不能預約 13、14、15、16，也不能預約 16 和 19，因爲後兩者都會違背至多三個時段且連續的限制。若使用者決定預約 2023/10/12 的 13、14、15 三個時段，則需選擇活動預約日期 2023/10/12、開始時間 13 與時間長度 3。

1.1.2 給 User 的功能

在本系統中，User 可以執行以下功能：

1. 新增讀書會：使用者能透過設定參與人數上限以及輸入讀書會內容、想討論的課程名稱等相關資訊來發起一場讀書會，一旦發起讀書會，系統便會給定一個屬於該讀書會的編號，接著會列出目前可租借的教室資訊及可用的日期及時段供使用者選擇。待使用者選擇並確定後，系統會將這筆活動新增到資料庫。
2. 查詢可加入的讀書會：使用者可查詢尚未結束且可參與的讀書會。
3. 參加讀書會：使用者可以透過輸入可加入的讀書會編號來參加讀書會。
4. 退出讀書會：使用者如果臨時不想參加，可退出已參與但未結束的讀書會。
5. 查詢使用者曾經舉辦過的讀書會：使用者可以查詢自身曾建立過的讀書會，包括已結束與正在進行中的。
6. 查詢使用者曾經參與過的讀書會：使用者可以查詢自身曾參與過的讀書會，包括已結束與正在進行中的。
7. 查詢課程：使用者可透過課程名稱或授課教師姓名來查詢課程。
8. 查詢指定日期下教室已被預約的時段：使用者若想得知某間教室在特定日期，可被預約或已被預約的時段。
9. 修改個人資料：使用者可以更新自己的姓名、密碼及 email。

[2]實際可供預約時段的範圍在未來可能視情況調整。

1.1.3 給 Admin 的功能

在本系統中，Admin 可以執行以下功能：

1. 管理課程：管理員可對課程資訊進行增刪改查的操作。
2. 管理教室：管理員可對租借教室的資訊進行增刪改查的操作。
3. 查詢使用者資訊：管理員可查詢所有使用者的活動紀錄，包括該使用者曾經發起及參與的讀書會有哪些。
4. 查詢讀書會資訊：管理員可查詢這些讀書會的詳細資訊。
5. 修改使用者資料：管理員可以更新任意 User 的姓名、密碼及 email。

1.1.4 系統級指令

由系統本身運行的功能如下：

1. 更新讀書會狀態：系統每隔一段時間會將超過讀書會舉辦時間的讀書會狀態從進行中（Ongoing），更新爲已結束（Finished）。
2. 確認使用者選擇的教室是否在指定日期與時段內已被預約：系統在使用者新增讀書會時，會判斷使用者所選擇租借的教室跟該日期與時段是否已被預約，若被預約回傳 1 ，沒有預約則回傳 0。

2 系統設計

2.1 ER Diagram

圖 1 是「I'm in」的 ER Diagram，在這個 ERD 中共有四個實體（entity），分別是 `USER`、`STUDY_EVENT`、`CLASSROOM`、`COURSE`，以及四個關係（relationship），包括 `HOLD`、`PARTICIPATE`、`BOOK`、`STUDY_FOR`。其中 `USER` 代表的是使用「I'm in」平台的任何人，任何人都須註冊才能開始使用。在註冊時，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代號及定義他在「I'm in」中的身份爲 User。經過後臺手動設定，可以將特定使用者的身份修改爲 Admin，此時該使用者便可以使用 User 和 Admin 的功能。

`STUDY_EVENT` 代表讀書會，使用者可以參加或發起多場讀書會。若想參加讀書會，系統會記錄使用者加入哪場讀書會及報名參與的時間點。而若想發起一場讀書會，則系統會需要使用者提供該場讀書會的資訊，包括讀書會的內容、參與人數上限，以及想討論的是哪堂課的課程等資訊。其中參與人數上限及課程資訊是必填資訊；而讀書會內容則是選填，使用者可依個人

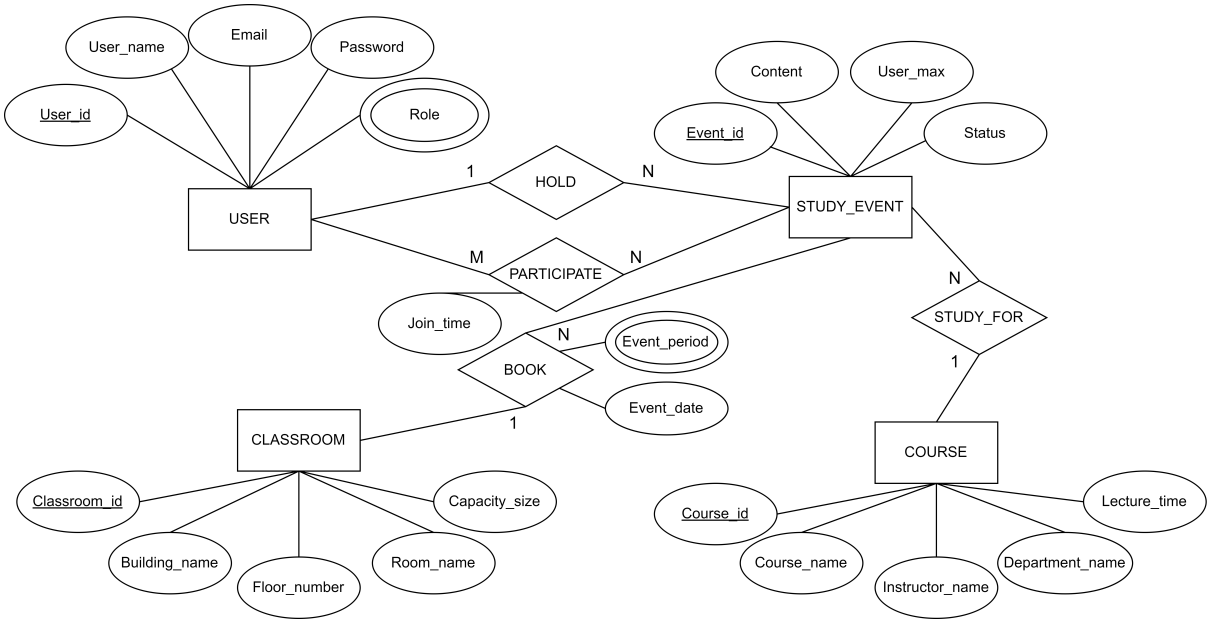

圖 1: 「I'm in」的 ER Diagram

需求選擇提供更多有關讀書會的描述。接著選擇在讀書會進行期間想租借的教室及確定的日期與時段。而系統也會記錄該讀書會的狀態是已結束或正在進行中。

系統會在使用者發起讀書會時列出目前可用的教室資訊，其中教室用 `CLASSROOM` 來表示，會呈現教室代號、教室所在建物名稱、所在樓層及該教室可以容納的人數上限這些資訊。其中教室所在樓層這項資訊視情況提供，若系統無法確定此資訊則會略過。當使用者確定租借某間教室後，系統會記錄租借的日期及時段，因爲讀書會是一次性的活動，所以一場讀書會被發起時，只能租借一間教室的某個特定日期，但可預約該日期的至多三個連續時段，而一間教室可以允許被多個讀書會給租借。

使用者在發起讀書會時，會有想討論的課程，透過 `COURSE` 來表示。系統會記錄課程代號、課程名稱、該課程的授課教師姓名、開設對象（該課程所屬的系所名稱），以及該課程時間。其中開設對象這項資訊視情況提供，若系統無法確定此資訊則會略過。一個讀書會的成立目的只能以討論單一課程爲主，不存在一個讀書會爲討論多堂課程內容的情況。

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 Diagram，一共有七個關聯（relation），分別是 `USER`、`USER_ROLE`、`STUDY_EVENT`、`STUDY_EVENT_PERIOD`、`PARTICIPATION`、`COURSE`、`CLASSROOM`。

`USER` 這個關聯的主鍵（Primary key，PK）是 `User_id`，該屬性被定義爲系統自動增加

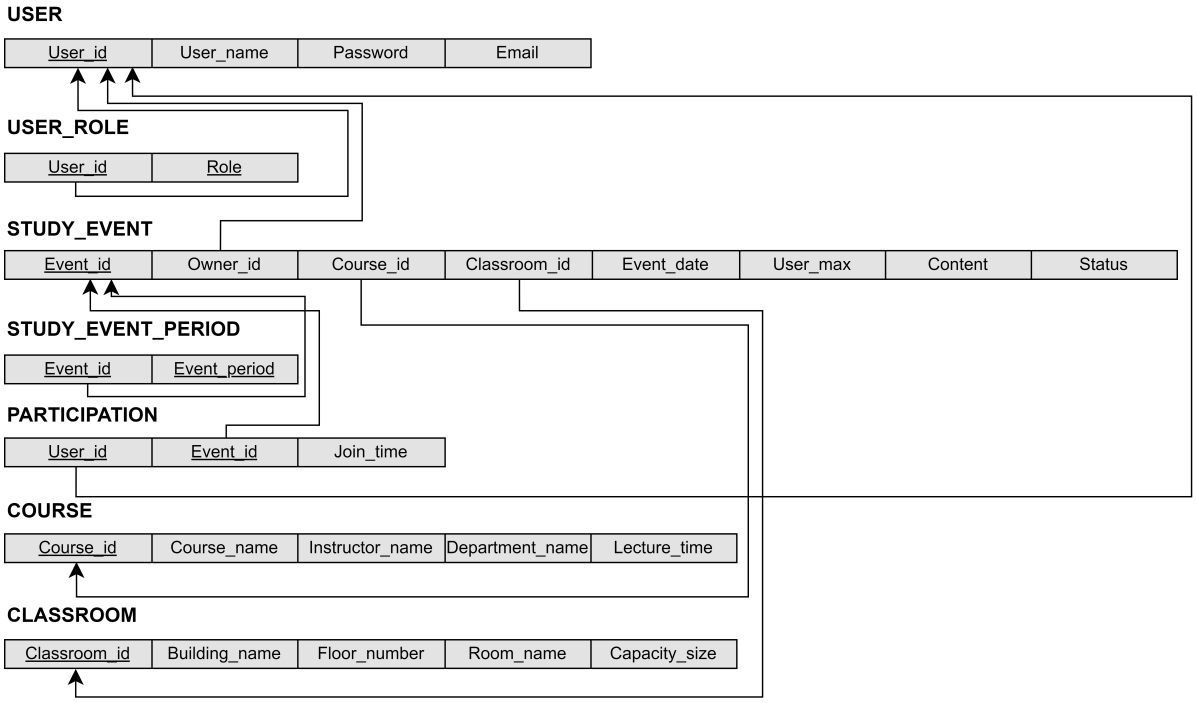

圖 2: 「I'm in」的 Relational Database Schema Diagram

的唯一識別碼，亦即每當有新的使用者被建立時，系統會自動爲 User_id 分配一個唯一的值，並且這個值起始值被設定爲 1，按照遞增的方式生成，以確保每個使用者都有一個唯一的識別碼。而 USER_ROLE 這個關聯則是由 USER 的 Role 延伸而來，因爲其作爲多值屬性，所以必須獨立出來成爲一個關聯，而該關聯的主鍵是由所有屬性組成，因此原隸屬於 USER 的 Role 也作爲 USER_ROLE 的主鍵之一，其中 User_id 同時也作爲外部鍵（Foreign key，FK）參考到 USER 的主鍵 User_id。

STUDY_EVENT 這個關聯的主鍵是 Event_id，同樣的該屬性也是被定義爲系統自動增加的唯一識別碼，亦即每當有新的讀書會紀錄被插入時，系統會自動爲 Event_id 分配一個唯一的值，並且這個值起始值被設定爲 1，按照遞增的方式生成，以確保每場讀書會都有一個唯一的識別碼，同理其他資料表的 ID 屬性都是透過這種方式生成，後續就不多做說明。外部鍵則包括參考到 USER 主鍵的 Owner_id、參考到 COURSE 主鍵的 Course_id，以及參考到 CLASSROOM 主鍵的 Classroom_id。而由於 STUDY_EVENT 的欄位 Event_period 是多值屬性，所以我們將 STUDY_EVENT_PERIOD 獨立出來成爲一個關聯，而該關聯的主鍵是由所有屬性（Event_id 和 Event_period）組成，同時 Event_id 也做爲外部鍵參考到 STUDY_EVENT 的主鍵。

PARTICIPATION 這個關聯是由多對多之 PARTICIPATE 關係型態產生而來，該關聯的主鍵由兩個外部鍵組合而成，分別作爲參考到連結該關係的兩個實體 USER 的主鍵 User_id 以及 STUDY_EVENT 的主鍵 Event_id。在我們的系統中，讀書會的創建者不算在參與者內，所以沒有參加的紀錄，因此 PARTICIPATION 這張表內針對一個讀書會不會記錄創建者的 ID，只會記錄非創建者的參與者的 ID。

COURSE 這個關聯的主鍵是 Course_id，一樣是被定義爲系統自動增加的唯一識別碼，剩餘的屬性則包含教室名稱、授課教師姓名、開設對象及課程時間這些 COURSE 實體下的屬性。最後是 CLASSROOM 關聯，該關聯的主鍵是 Classroom_id，被定義爲系統自動增加的唯一識別碼，其餘屬性則爲 CLASSROOM 實體下的屬性包括教室所在的建物名稱、教室所在樓層、教室名稱及教室容納人數上限。

2.2.2 考量效能而做的設計變更

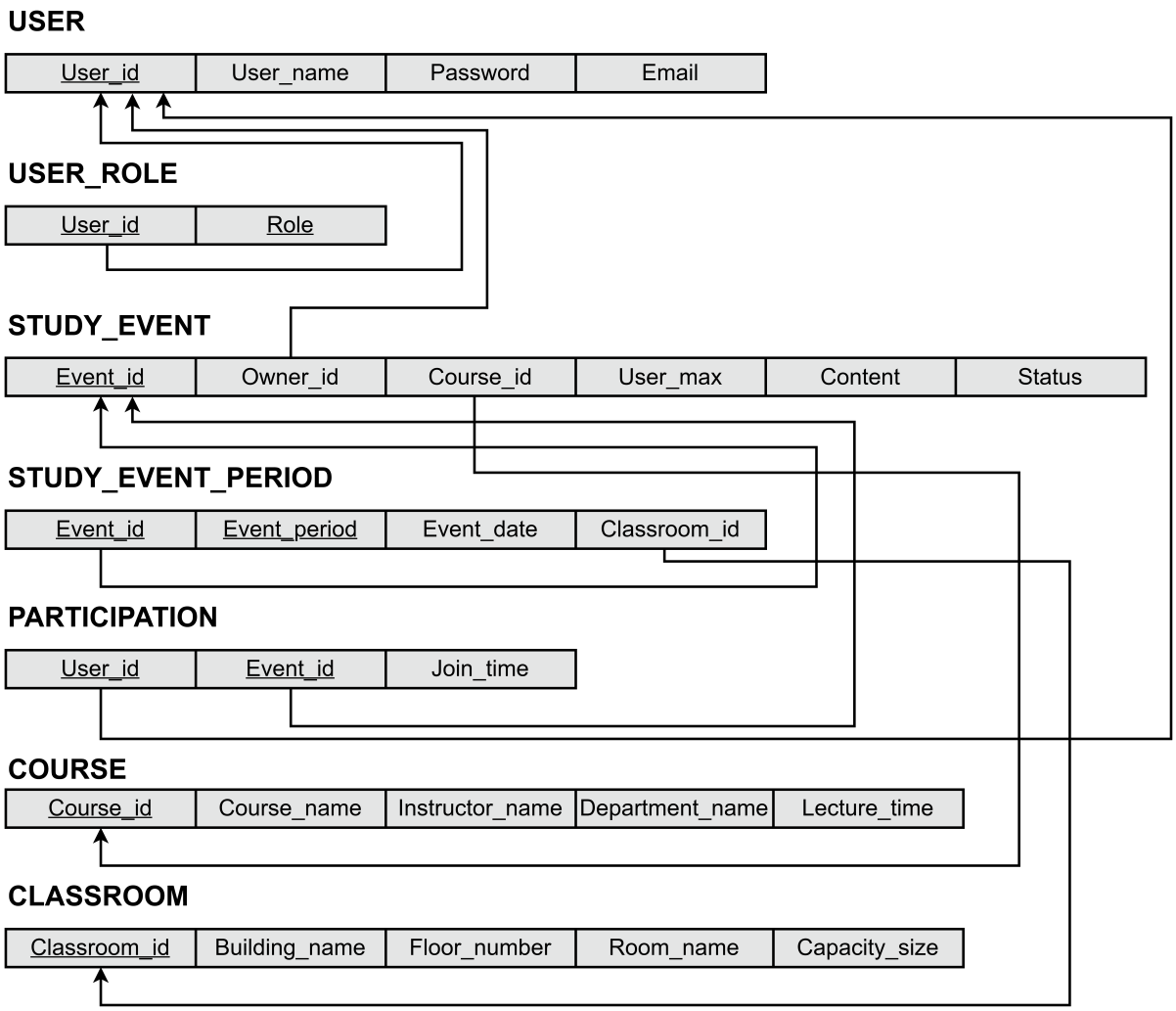

圖 3: 改良過的「I'm in」的 Relational Database Schema Diagram

原本 STUDY_EVENT 及 STUDY_EVENT_PERIOD 兩個關聯應該如 2.2.1 所描述含有那些欄位。然而儘管如此，在考量到我們系統的特定功能會特別關注查詢某個教室被預約的情況需求。基於這一需求，我們稍微地調整了一下這兩個關聯的欄位。如圖 3 所示，將原先在 STUDY_EVENT 關聯中的 Classroom_id 教室代號及 Event_date 讀書會舉辦日期這兩個欄位移至 STUDY_EVENT_PERIOD 關聯。將所有與預約相關的欄位併在一起，放在

STUDY_EVENT_PERIOD 這個用來儲存讀書會預約紀錄的關聯，並由 Event_id 及 Event_period 組成複合主鍵。我們認爲透過這樣的調整，可以減少查詢時的 JOIN 操作，從而簡化了特定查詢的複雜性並提高了效率。這一設計決策是出於對查詢性能的優化考慮，通過將相關的資訊集中儲存，能更輕鬆地檢索某個教室的預約情況，而無須涉及 STUDY_EVENT 表。儘管這種設計可能會稍微違反正規化，但在當前的應用考量，我們認爲這種權衡是合理的，以便更好地滿足特定的業務需求。

2.3 Data Dictionary

「I'm in」的資料表共有圖 3 所示的七個，各個資料表的欄位相關資訊依序呈現在表 1 到表 7。

Column Name	Meaning	Data Type	Key	Constraint	Domain
User_id	使用者代號	bigint	PK	Not Null	AUTO_INCREMENT
User_name	使用者名稱	varchar (50)		Not Null	
Password	使用者密碼	varchar (15)		Not Null	
Email	使用者信箱	varchar (50)		Not Null, Unique	

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	{User, Admin}
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
Event_id	讀書會代號	bigint	PK	Not Null	AUTO_INCREMENT
Owner_id	創建者代號	bigint	FK: USER(User_id)	Not Null	
Course_id	課程代號	bigint	FK: COURSE(Course_id)	Not Null	
User_max	參與人數上限	int		Not Null	
Content	讀書會內容	varchar (100)			
Status	讀書會狀態	varchar (10)		Not Null	{Ongoing, Finished}
Referential triggers		On Delete	On Update		
Owner_id: USER(User_id)		Cascade	Cascade		
Course_id: COURSE(Course_id)		Cascade	Cascade		

表 3: 資料表 STUDY_EVENT 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
Event_id	讀書會代號	bigint	PK, FK: STUDY_EVENT(Event_id)	Not Null	
Classroom_id	教室代號	bigint	FK: CLASSROOM(Classroom_id)	Not Null	
Event_date	讀書會舉辦的日期	date		Not Null	
Event_period	讀書會舉辦的時段	int	PK	Not Null	8–21
Referential triggers		On Delete	On Update		
Event_id: STUDY_EVENT(Event_id)		Cascade	Cascade		
Classroom_id: CLASSROOM(Classroom_id)		Cascade	Cascade		

表 4: 資料表 STUDY_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	讀書會代號	bigint	PK, FK: STUDY_EVENT(Event_id)	Not Null	
Join_time	報名參與時間	timestamp		Not Null	
Referential triggers		On Delete	On Update		
User_id: USER(User_id)		Cascade	Cascade		
Event_id: STUDY_EVENT(Event_id)		Cascade	Cascade		

表 5: 資料表 PARTICIPATION 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
Course_id	課程代號	bigint	PK	Not Null	AUTO_INCREMENT
Course_name	課程名稱	varchar (50)		Not Null	
Instructor_name	授課教師姓名	varchar (50)		Not Null	
Department_name	開設對象	varchar (20)			
Lecture_time	課程時間	varchar (20)		Not Null	

表 6: 資料表 COURSE 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
Classroom_id	教室代號	bigint	PK	Not Null	AUTO_INCREMENT
Building_name	教室所在建物名稱	varchar (30)		Not Null	
Floor_number	教室所在樓層	int			
Room_name	教室名稱	varchar (20)		Not Null	
Capacity_size	教室容納人數上限	int		Not Null	

表 7: 資料表 CLASSROOM 的欄位資訊

2.4 正規化分析

當設計關聯式資料庫時，我們可以檢視資料庫綱目（database schema）是否滿足正規化（normalization）條件，因此我們將依序從第一正規式（1NF）到第四正規式（4NF）來說明「I'm in」的關聯是如何滿足這些規則。

在 1NF 方面，如果每個關聯的屬性都是 simple 且 single-valued，換句話說，在關聯中沒有任何一個屬性是 composite 或 multi-valued，則滿足 1NF。在第 2.2 節中，我們說明了如何將 multi-valued 屬性 `Role` 及 `Event_period` 從原關聯中獨立出來，以獨立的關聯描述，我們的 schema 因此滿足 1NF。

在 2NF 方面，如果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（candidate key），也就是沒有出現部分功能相依性（partial functional dependency），則滿足 2NF。在第 2.2.2 節中，我們爲了減少查詢時的 JOIN 操作，將原本放在 `STUDY_EVENT` 關聯的兩個屬性 `Classroom_id` 及 `Event_date` 移到 `STUDY_EVENT_PERIOD` 關聯中，並維持以 `Event_id` 及 `Event_period` 做爲複合主鍵，此修改會使 {`Event_id`} → {`Classroom_id`, `Event_date`} 的部分功能相依性存在，因此我們的設計違反 2NF。儘管如此，誠如第 2.2.2 節中所討論的，我們認爲在當前的應用考量，爲了查詢性能的優化，些微地不滿足正規化設計，這種權衡是合理的，以便更好地滿足特定的業務需求。

在 3NF 方面，如果一個關聯中的非鍵屬性都沒有遞移相依（transitively dependency）於主鍵，則滿足 3NF。檢視設計的關聯，沒有存在任何遞移相依，因此符合 3NF。在比 3NF 更嚴格的 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），也就是要確保 $X \to Y$ 的 X 一定是超級鍵，而我們的 schema 也符合 BCNF。

最後是 4NF，由於「I'm in」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 `COURSE`、`CLASSROOM` 資料表內的資料建置方式，我們首先利用爬蟲從台大課程網收集 901 筆教室資料與 8,497 筆課程資料，並匯入至資料庫對應的教室與課程資料表。由於部分爬蟲後得到的教室資料缺乏教室所在樓層 `Floor_number` ，因此該欄位可爲空值。課程則因爲如通識課、體育課等這類課程缺乏開設對象 `Department_name`，因此該欄位也可爲空值。

`USER` 資料表內的資料建置方式，則採取隨機生成，共生成 4,000 個使用者。接著，我們利用這些使用者模擬使用者新增讀書會的行爲，隨機挑選讀書會討論的課程等其他資訊，創建一筆又一筆的讀書會紀錄，儲存在 `STUDY_EVENT` 資料表，共有 30,000 筆讀書會資料，接著會爲那些讀書會隨機預約教室與讀書會舉辦的日期與時段，其中創建時會確認挑選的時段是否有重疊，若有則會捨棄該筆資料，最終共生成出 788,845 筆讀書會預約記錄的資料，存在

STUDY_EVENT_PERIOD 資料表中。

3.2 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 Admin 的功能。在第 3.2.1 和第 3.2.2 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 3.2.3 小節中，我們也列出系統級指令其對應的 SQL 指令。

3.2.1 給 User 的功能

1. 新增讀書會：若要實現此功能，假設情境爲「創建者代號 Owner_id 『4098』想新增一筆讀書會內容 Content 爲『讀期中考』，讀書會狀態 Status 設定爲『Ongoing』，讀書會的參與人數上限 User_max 則設定爲『20』，想討論的課程名稱 Course_name 爲『線性代數』，對應的課程編號 Course_id 是『20』，並預約教室代號 Classroom_id 爲『1』的教室，且日期 Event_date 是『2023-11-12』，舉辦的時段 Event_period 則是『13、14、15』這三個時段。」則對應的 SQL 指令如下。系統會在 STUDY_EVENT 的資料表新增一筆讀書會的資料，並在 STUDY_EVENT_PERIOD 資料表中爲該讀書會新增三筆預約紀錄，每個時段一筆，共新增三筆。

```
Insert Into STUDY_EVENT (Content, Status, User_max, Course_id,
                         User_id)
Values ('讀期中考', 'Ongoing', 20, 4033, 4098);
Set @newID = last_insert_id();
Insert Into STUDY_EVENT_PERIOD (Event_date, Event_period,
                                Classroom_id, Event_id)
Values
('2024-11-12', 13, 1, @newID),
('2024-11-12', 14, 1, @newID),
('2024-11-12', 15, 1, @newID);
```

Listing 1: 新增讀書會 SQL 指令

2. 查詢可加入的讀書會：若要實現此功能，查詢條件爲「讀書會狀態 Status 爲『進行中（Ongoing）』且目前參與人數小於該讀書會的參與人數上限 User_max。」對應 SQL 指令如下。系統會執行該查詢，並回傳尚未結束仍可加入的讀書會資訊供使用者選擇。值得注意的是，由於在我們的系統中 PARTICIPATION 這張表內不會記錄讀書會創建者的 ID，只會記錄參與者的 ID，所以如果有一個讀書會還沒有任何人參與，該讀書會的 ID 就不會出現在 PARTICIPATION 這張表中。這是爲什麼這裡我們使用的是 Left Join 而非 Join，讓還沒有任何人參與的讀書會也能被列在回傳結果中。

```sql
Select se.*
From STUDY_EVENT As se
    Left Join PARTICIPATION As p On se.Event_id = p.Event_id
Where se.Status = 'Ongoing'
Group By se.Event_id
Having Count(p.User_id) <
(
    Select User_max
    From STUDY_EVENT AS se2
    Where se.Event_id = se2.Event_id
);
```

Listing 2: 查詢可加入的讀書會 SQL 指令

3. 參加讀書會：若要實現此功能，假設情境爲「創建者代號 Owner_id 『4098』想參加讀書會編號 Event_id 『1348』的讀書會，並於 join_time 『2024-10-24 00:00:00』提出申請。若該讀書會爲可加入的，系統會將該使用者加入讀書會，並記錄加入時間；否則，系統退回此申請。

```sql
Insert Into PARTICIPATION (User_id, Event_id, Join_Time)
Values (4098, 1348, 2024-10-24 00:00:00);
```

Listing 3: 參加讀書會 SQL 指令

4. 退出讀書會：若要實現此功能，假設情境爲「使用者代號 User_id 『2』想要退出讀書會代號 Event_id 爲『267408』的讀書會。」則對應的 SQL 指令如下。系統會在 PARTICIPATION 資料表中刪除該使用者特定的讀書會參與紀錄。

```sql
Delete From PARTICIPATION
Where Event_id = 267408 And User_id = 2;
```

Listing 4: 退出讀書會 SQL 指令

5. 查詢使用者曾經參與過的讀書會：若要實現此功能，假設情境爲「使用者代號 User_id 『2』想要查看自己曾經參與過的所有讀書會。」則對應的 SQL 指令如下。系統會執行該查詢，並回傳包含使用者代號爲 2 的使用者曾經參與的所有讀書會的相關資訊。

```sql
Select se.*
From PARTICIPATION As p
    Join STUDY_EVENT As se On p.Event_id = se.Event_id;
```

```
Where p.User_id = 2;
```

Listing 5: 查詢使用者曾經參與過的讀書會 SQL 指令

6. 查詢課程：若要實現此功能，假設情境爲「某使用者想透過兩個條件，授課教師姓名 Instructor_name 內有『孔』或課程名稱 Course_name 內有『資料庫』來查詢課程。」則對應的 SQL 指令如下，系統會執行該查詢，並回傳符合這些條件的課程資訊。

```
Select *
From COURSE
Where Instructor_name Like '%孔%'
Or Course_name Like '%資料庫%';
```

Listing 6: 查詢課程 SQL 指令

7. 查詢指定日期下教室已被預約的時段：若要實現此功能，假設情境爲「某使用者想查詢日期爲『2023-11-10』已被預約的教室以及時段」則對應的 SQL 指令如下。系統會執行該查詢，並回傳包含教室名稱和時段的結果，這些教室和時段已在指定日期被預約。

```
Select c.Room_name, sep.Event_period
From STUDY_EVENT_PERIOD As sep
    Join CLASSROOM As c On sep.Classroom_id = c.Classroom_id
Where sep.Event_date = '2023-11-10';
```

Listing 7: 查詢指定日期下教室已被預約的時段 SQL 指令

3.2.2 給 Admin 的功能

1. 管理教室（增刪改查）
   新增教室：若要實現此功能，假設情境爲「某管理員想要新增一筆教室所在建物名稱 Building_name 爲『共同』，教室容納人數上限 Capacity_size 爲『120』，教室所在樓層 Floor_number 爲『3』，教室名稱 Classroom_name 爲『312』的教室。」則對應的 SQL 指令如下。系統會在 CLASSROOM 資料表中新增一筆教室資訊。

```
Insert Into CLASSROOM(Building_name, Capacity_size,
                      Floor_number, Room_name)
Values ('共同', 120, 3, '312');
```

Listing 8: 新增教室 SQL 指令

刪除教室：若要實現此功能，假設情境爲「某管理員想刪除代號 Classroom_id 爲『903』的教室資訊。」則對應的 SQL 指令如下。系統會在 CLASSROOM 資料表中刪除該教室的資訊。

```
Delete From CLASSROOM
Where Classroom_id = 903;
```

Listing 9: 刪除教室 SQL 指令

更新教室：若要實現此功能，假設情境爲「某管理員想要更改教室代號 Classroom_id 爲『901』的教室資訊，將其教室容納人數上限 Capacity_size 改爲『10』。」則對應的 SQL 指令如下。系統會更新 CLASSROOM 資料表中，教室代號及容納人數上限兩個欄位資訊。

```
Update CLASSROOM
Set Capacity_size = 10
Where Classroom_id = 901;
```

Listing 10: 更新教室 SQL 指令

查詢教室：若要實現此功能，假設情境爲「某管理員想要查詢教室所在建物名稱 Building_name 爲『共同』的所有教室。」則對應的 SQL 指令如下。系統會執行該查詢，並回傳符合條件的教室資訊。

```
Select *
From CLASSROOM
Where Building_name = '共同';
```

Listing 11: 查詢教室 SQL 指令

2. 查詢使用者資訊：若要實現此功能，假設情境爲「某管理員想要查詢使用者代號 User_id 爲『1』的使用者資訊。」則對應的 SQL 指令如下。系統會執行該查詢，並回傳特定使用者的相關資訊。

```
Select *
From USER
Where User_id = 1;
```

Listing 12: 查詢使用者資訊 SQL 指令

3. 查詢讀書會資訊：若要實現此功能，假設情境爲「某管理員想要查詢讀書會討論的課程名稱 Course_name 爲『線性代數』的讀書會資訊。」則對應的 SQL 指令如下。系統會執行該查詢，並回傳特定讀書會的相關資訊。

```sql
Select se.*
From STUDY_EVENT As se
    Join COURSE As c On se.Course_id = c.Course_id
Where c.Course_name = '線性代數';
```

Listing 13: 查詢讀書會資訊 SQL 指令

3.2.3 系統級指令

1. 更新讀書會狀態：若要實現此功能，假設情境爲「系統將讀書會舉辦的日期 Event_date 爲『2023-10-30』且舉辦時段 Event_period 爲『9』的所有讀書會狀態 Status 更改爲『已結束（Finished）』。」則對應的 SQL 指令如下。系統會更新 STUDY_EVENT 表中特定讀書會的讀書會狀態欄位。

```sql
Update STUDY_EVENT As se
    Join STUDY_EVENT_PERIOD As sep On se.Event_id = sep.Event_id
Set se.Status = 'Finished'
Where se.Status='Ongoing'
And sep.Event_date = '2023-10-30'
And sep.Event_period = 9;
```

Listing 14: 更新讀書會狀態 SQL 指令

2. 確認使用者選擇的教室是否在指定日期與時段內已被預約：若要實現此功能，假設情境爲「查詢教室代號 Classroom_id 爲『1』的教室在日期『2023-11-1』的時段『10-13』之間是否已被預約，若被預約回傳 1（True），若沒有預約回傳 0（False）。」則對應的 SQL 指令如下。

```sql
Select
Case
    When Exists
    (
        Select *
        From STUDY_EVENT_PERIOD As sep
        Where sep.Classroom_id = 1
        And sep.Event_date = '2023-11-1'
        And sep.Event_period >= 10
        And sep.Event_period <= 13
    )
```

```
    Then 1
    Else 0
End;
```

Listing 15: 確認教室是否被預約 SQL 指令

3.3 SQL 指令效能優化與索引建立分析

3.3.1 在 STUDY_EVENT 中加入讀書會狀態的索引

我們認爲「I'm in」系統中最常使用到的功能爲「查詢可加入的讀書會」，因此我們決定針對該指令進行效能優化。如 3.2.1 所示，讀書會是否可加入需要判斷「讀書會是否爲進行中」。由於讀書會狀態 `Status` 原本是雜亂地分佈於資料庫之中，理論上透過建立索引，我們應該能更有效率地找到進行中的讀書會。

爲了優化效能，我們爲 `STUDY_EVENT` 中的「讀書會的參與人數上限 `user_max`」建立索引，語法如下。

```
Create Index idx_study_event_status
On STUDY_EVENT(status)
```

Listing 16: 建立讀書會狀態索引

3.3.2 在 STUDY_EVENT_PERIOD 中加入讀書會舉辦日期、教室代號索引

由於「I'm in」系統中 `STUDY_EVENT_PERIOD` 資料表中的資料量最大，該資料表是用來儲存讀書會的預約紀錄，無論是使用者要實現功能「查詢教室可預約的時間」或是系統級指令「確認使用者選擇的教室與時間是否已被預約」及「更新讀書會狀態爲已結束」，這些需要以讀書會舉辦的日期爲條件來搜尋該資料表的操作，執行時間都會較費時。除此之外，使用者要實現功能「查詢教室可預約的時間」或是系統級指令「確認使用者選擇的教室與時間是否已被預約」時，也常常使用到「教室代號」的資訊。

爲了能有效優化效能，我們爲該資料表中的「讀書會舉辦的日期 `Event_date`」及教室代號 `Classroom_id` 欄位建立索引，語法如下。

```
CREATE INDEX idx_event_date_classroom
ON STUDY_EVENT_PERIOD(event_date, classroom_id)
```

Listing 17: 建立讀書會舉辦日期及教室代號索引

3.3.3 索引效果

對於 3.3.1 中我們設計的索引，我們利用第 3.2.1 小節中「查詢可加入的讀書會」所使用的 SQL 指令來測試索引是否有成效，又帶來多大程度的效能提升。建立索引前運行五次的結果分別是 1.373 秒、0.793 秒、0.706 秒、0.759 秒、0.756 秒，平均運行時間約爲 0.8774 秒，標準差約爲 0.2493 秒；建立索引後運行五次的結果分別是 0.81 秒、0.844 秒、0.766 秒、0.716 秒、0.698 秒，平均運行時間約爲 0.7668 秒，標準差約爲 0.0550 秒。

對於 3.3.2 中我們設計的索引，我們利用第 3.2.1 小節中「查詢指定日期下教室已被預約的時段」所使用的 SQL 指令來測試索引是否有成效，又帶來多大程度的效能提升。建立索引前運行五次的結果分別是 0.113 秒、0.092 秒、0.067 秒、0.066 秒、0.092 秒，平均運行時間約爲 0.0860 秒，標準差約爲 0.0177 秒；建立索引後運行五次的結果分別是 0.078 秒、0.067 秒、0.062 秒、0.076 秒、0.070 秒，平均運行時間約爲 0.0706 秒，標準差約爲 0.00585 秒。

在這兩個例子中，建立索引後平均運行時間的降低顯示了索引對查詢效率產生了顯著的影響。儘管如此，爲了評估索引的效果，可能需要更多的測試與分析，才能更全面地了解索引的實際效果。

3.4 交易管理

在給 Admin 功能中管理課程的部分，我們另外提供一個批次匯入課程資訊的功能。該功能可允許管理員上傳包含課程資訊的 CSV 檔，系統則會將檔案內的每一筆課程分別寫入資料庫。我們使用 `psycopg2` 套件實現此功能，在 `./action/course_management/UploadCourses.py` 中，從 client 端接收 CSV 檔案後，便會呼叫 `./DB_util.py` 中的 `upload_courses` 函式。由於 `psycopg2` 在預設情況下會自動開始交易，因此不需要手動執行 `BEGIN` 相關操作，程式會直接針對每一筆資料進行 `INSERT`，在這個過程中如果出現違反資料表限制的情況，例如課程名稱不可爲空值等，新增的動作會立即停止。此時，系統將使用 `db.rollback()` 方法撤回該次交易（其中 `db` 爲 `psycopg2` 中的 `connection` 類別，作爲資料庫的連線），取消之前的所有資料庫異動。反之，若整個匯入過程順利完成且未發生任何資料一致性問題，系統將在最後使用 `db.commit()` 提交交易，確保所有的課程資訊都已成功儲存至資料庫。

3.5 併行控制

在我們的系統中，使用者發起讀書會活動後，會進入預約教室的流程，我們認爲這段流程是需要作併行控制的。首先，預約教室的流程如下：使用者會先透過系統查詢可預約時段，接著選擇他（她）想預約的時段，填入讀書會活動的資訊，然後發送新增請求。系統在這之後會檢查這些預約時段是否可被預約，如果可以，則儲存讀書會活動與預約資訊。

可以發現這段流程很有必要加鎖，不然就可能會發生使用者 A 查詢可預約時段後發現某教室某時段可以被借用，接著開始做選擇、填入資訊、新增借用等動作，於此同時使用者 B 也查詢並且發現該教室該時段可以被借用，然後做後續的借用。如果沒有靠著加鎖做併行控制，

該教室在該時段就有可能被借給複數位使用者，亦即發生重複借用這種不該發生的情況。若以我們的資料庫來說，就是 STUDY_EVENT_PERIOD 表中最終會有多筆資料，它們的 Event_id、Event_date 和 Event_period 是不同的，但 Classroom_id 相同。

我們認爲有兩種適合的做法都可以避免發生重複借用。第一種是在查詢可預約時段前開始加鎖，防止其他交易在查詢時段到借用的期間同時進行預約或修改相關資訊，並在預約完成或取消後釋放鎖。透過這種作法，可以確保該教室的該時段在被完成預約前一直處於上鎖狀態。第二種是在使用者按下「確認預約」後開始加鎖，此時系統再次查詢檢查該教室在預計被預約的時段是否是未被借用的，若是則新增借用記錄到 STUDY_EVENT_PERIOD 並且解鎖，若否則不新增並且解鎖然後在螢幕上顯示錯誤訊息。如此亦可確保不會被重複借用。

由於在使用者真正按下「確認預約」並發送新增請求前，可能存在過長的等待時間，因此我們選擇將併行控制的焦點放在發送新增請求後的流程，採用第二種做法，以避免不必要的等待並且傷害系統效能。爲實作此功能，我們使用 Python 的 threading 套件，在 ./DB_util.py 中的 create_study_group 函式裡，先透過 lock.acquire() 取得鎖後，再檢查該教室該時段是否已被借用，如果沒有則新增預約紀錄至資料庫中並透過 lock.release() 解鎖，讓其他連線可以使用此功能；若已被借用，則解鎖並回傳借閱失敗。

4 分工資訊

這部分就不提供同學們示範，請參照期末專案任務敘述文件「報告中應簡明清楚地描述每一位組員負責專案中的哪些項目。」的要求自行撰寫。

5 專案心得

這部分同樣不提供同學們示範，請參照期末專案任務敘述文件「每個組員都應個別寫下一段簡單的心得感想，去描述自己在專案中學到的東西、體驗到的困難，以及其他任何想說的話。」的要求自行撰寫。