資料庫管理（113-1）
期末專案報告

第 11 組
B11705044 江彥宏、B11705051 陳奕廷、B11705058 廖振翔
2024 年 12 月 10 日
GitHub 專案連結、展示影片連結

1 系統分析

在現今數位化的時代，購票系統扮演著連接活動主辦方與參與者的重要角色。傳統的購票方式多依賴實體票券，存在著排隊購票、票券損壞、資訊更新不及時等問題。因此，我們設計並實作了一個線上購票系統，旨在提升購票效率、便利性，並提供更完善的管理功能給活動主辦方。

本系統主要分為兩類使用者：一般使用者（User）與管理員（Admin）。一般使用者可以瀏覽活動、購買票券、取消訂單、查看活動詳情、進行付款、查看購票歷史以及編輯個人資訊。管理員則擁有更高的權限，可以新增活動、發行票券、查詢使用者資訊、查看使用者購票歷史及管理使用者資料等功能。

1.1 系統功能

1.1.1 給 User 的功能

在本系統中，User 可以執行以下功能：

1. 註冊帳號：使用者可透過提供姓名、電子郵件及密碼來註冊新帳號，註冊成功後可登入系統使用其他功能。
2. 登入系統：已註冊的使用者可透過電子郵件及密碼登入系統，進入個人專屬的操作介面。
3. 瀏覽活動：使用者可根據活動類別篩選感興趣的活動，並提供活動名稱、日期、地點及簡介等資訊。
4. 尋找活動：使用者可透過關鍵字尋找活動，快速找到目標活動。
5. 購買票券：使用者可選擇參與活動並購買相應的票券，系統會即時更新票券數量，避免超賣現象。
6. 取消訂單：若使用者改變計劃，可在活動開始前取消已購買的票券訂單，系統將進行退款處理。

7. 進行付款：使用者在購買票券時，可選擇不同的付款方式進行支付，確保交易的安全性。
8. 查看購票歷史：使用者可查看自己過去購買的所有票券訂單，包含訂單狀態、活動資訊及付款詳情。
9. 編輯個人資訊：使用者可更新自己的姓名、密碼及電子郵件等個人資料，保持資訊的最新性。

1.1.2 給 Admin 的功能

在本系統中，Admin 可以執行以下功能：

1. 新增活動：管理員可在系統中新增新活動，包含活動名稱、日期、地點、簡介及相關附件等資訊。
2. 發行票券：管理員可為每場活動發行不同類型的票券，設定票券價格、數量及其他相關屬性。
3. 查詢使用者資訊：管理員可查詢所有使用者的基本資訊，包含姓名、電子郵件及購票紀錄等。
4. 查看使用者購票歷史：管理員可查看特定使用者的所有購票訂單，了解其參與過的活動及消費行為。
5. 查看銷售報表：管理員可查看特定活動的銷售報表，了解該活動所發行的所有票券種類，銷售數量，盈餘以及總盈餘。

2 系統設計

2.1 ER Diagram

圖 1 是本售票系統的 ER Diagram，在這個 ERD 中共有九個實體（entity），分別是 `CUSTOMER`、`ORDER`、`ORDER_DETAIL`、`PAYMENT`、`TICKET`、`EVENT`、`ORGANIZER`、`VENUE`、`CATAGORY`。此外，系統中存在多個關係（relationship），包括 `SENT`、`PAID_FOR`、`WITH`、`INCLUDES_TICKET`、`SELL`、`LOCATED_AT`、`HOLD` 以及 `BELONG_TO`。

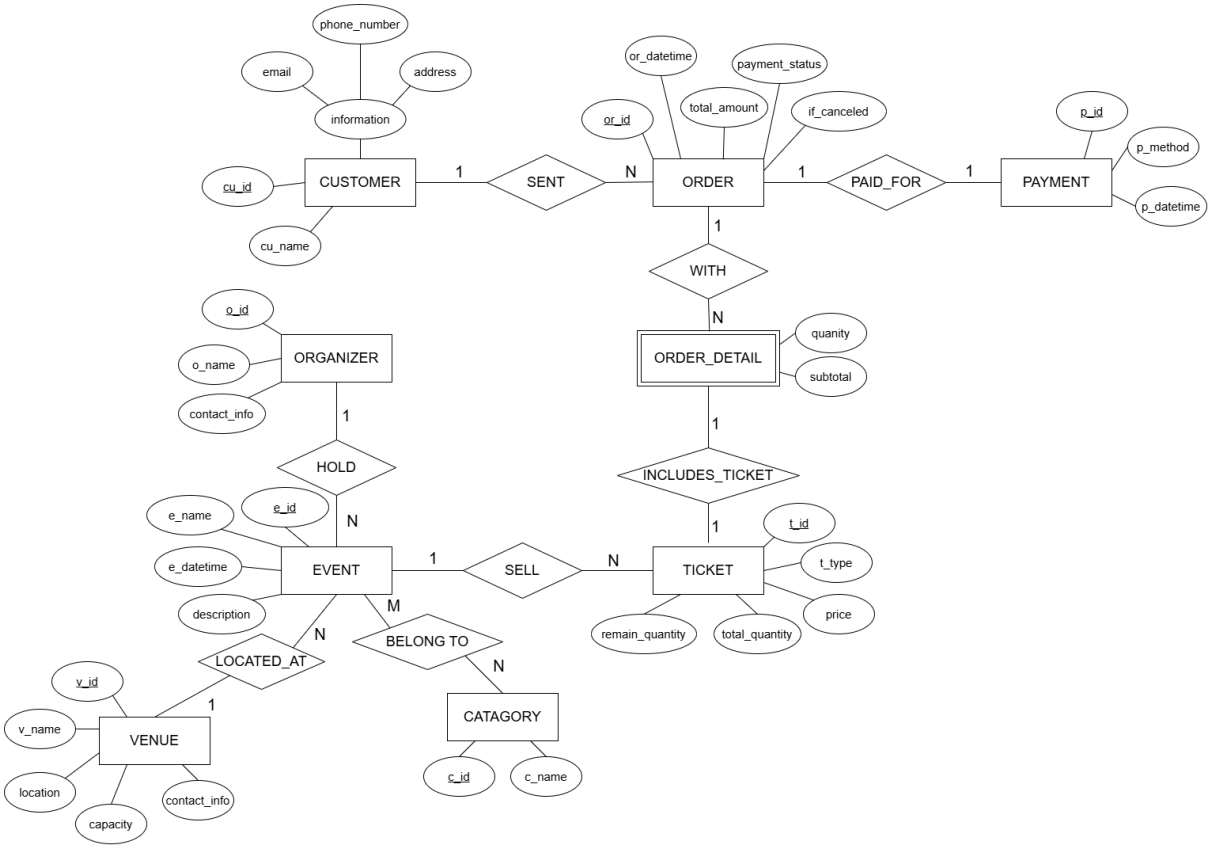

圖 1: 購票系統的 ER Diagram

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，一共由十個關聯 (relation) 組成，分別是 `CUSTOMER`、`ORDER`、`ORDER_DETAIL`、`PAYMENT`、`TICKET`、`ORGANIZER`、`EVENT`、`CATAGORY`、`VENUE`、`EVENT_VENUE`。

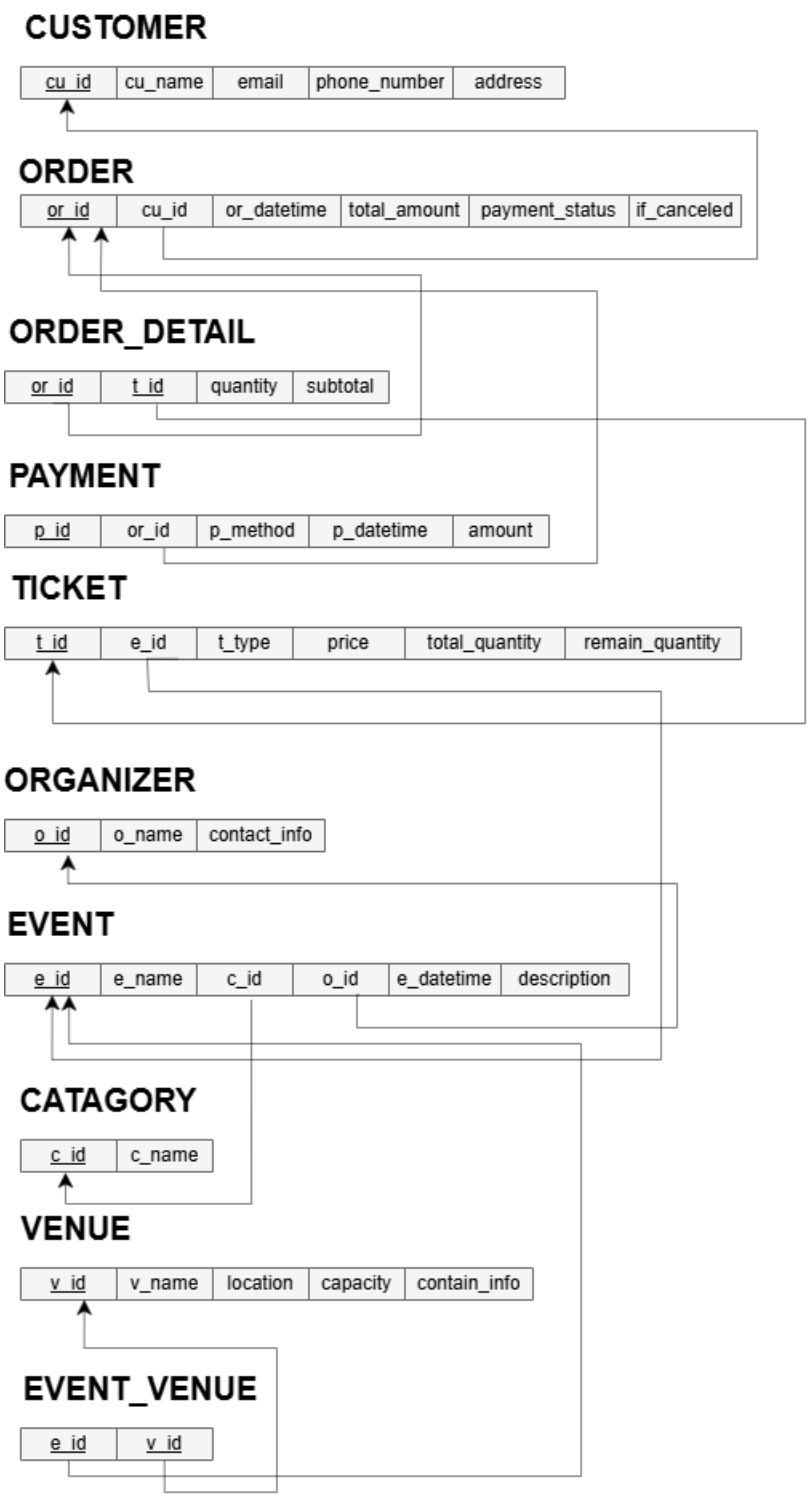

圖 2: 購票系統的 Relational Database Schema Diagram

2.3 Data Dictionary

「售票系統」的資料表共有圖 2 所示的十個，各個資料表的欄位相關資訊依序呈現在表 1 到表 10。

Column Name	Meaning	Data Type	Key	Constraint	Domain
c_id	類別 ID	INT	PK	Not Null, Unique	整數
c_name	類別名稱	VARCHAR(15)		Not Null	字串 (名稱)

表 1: 資料表 CATEGORY 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
e_id	活動 ID	INT	PK	Not Null, Unique	整數
e_name	活動名稱	VARCHAR(15)		Not Null	字串 (名稱)
c_id	類別 ID	INT	FK: Category(c_id)	Not Null	整數
o_id	主辦方 ID	INT	FK: Organizer(o_id)	Not Null	整數
e_datetime	活動日期及時間	DATETIME		Not Null	日期時間格式
e_location	活動地點	VARCHAR(15)		Not Null	字串 (地點)
description	活動描述	TEXT			字串 (描述)
Referential triggers		On Delete	On Update		
c_id: Category(c_id)		Cascade	Cascade		
o_id: Organizer(o_id)		Cascade	Cascade		

表 2: 資料表 EVENT 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
o_id	主辦方 ID	INT	PK	Not Null, Unique	整數
o_name	主辦方名稱	VARCHAR(15)		Not Null	字串 (名稱)
contact_info	主辦方聯絡資訊	VARCHAR(15)			字串 (聯絡資訊)

表 3: 資料表 ORGANIZER 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
t_id	票 ID	INT	PK	Not Null, Unique	整數
e_id	活動 ID	INT	FK: Event(e_id)	Not Null	整數
t_type	票種	VARCHAR(10)			字串 (票種)
price	票價	DECIMAL(10,2)		Not Null	正數
total_quantity	總票數	INT		Not Null	正整數
remain_quantity	剩餘票數	INT		Not Null	正整數
Referential triggers		On Delete	On Update		
e_id: Event(e_id)		Cascade	Cascade		

表 4: 資料表 TICKET 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
cu_id	客戶 ID	INT	PK	Not Null, Unique	整數
cu_name	客戶名稱	VARCHAR(15)		Not Null	字串 (名稱)
email	客戶電子郵件	VARCHAR(15)			字串 (電子郵件)
phone_number	客戶電話	VARCHAR(15)			字串 (電話號碼)
address	客戶地址	TEXT			字串 (地址)

表 5: 資料表 CUSTOMER 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
or_id	訂單 ID	INT	PK	Not Null, Unique	整數
cu_id	客戶 ID	INT	FK: Customer(cu_id)	Not Null	整數
or_date	訂單日期	DATE		Not Null	日期格式
total_amount	總訂單金額	DECIMAL(10,2)			正數
payment_status	付款狀態	VARCHAR(10)			字串 (狀態)
is_canceled	是否取消訂單	BOOLEAN		NOT NULL DEFAULT FALSE	TRUE 或 FALSE
Referential triggers		On Delete	On Update		
cu_id: Customer(cu_id)		Cascade	Cascade		

表 6: 資料表 ORDER 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
or_id	訂單 ID	INT	FK: Order(or_id)	Not Null	整數
t_id	票 ID	INT	FK: Ticket(t_id)	Not Null	整數
quantity	票數	INT		Not Null	正整數
subtotal	小計	DECIMAL(10,2)		Not Null	正數
Referential triggers		On Delete	On Update		
or_id: Order(or_id)		Cascade	Cascade		
t_id: Ticket(t_id)		Cascade	Cascade		

表 7: 資料表 ORDER_DETAIL 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
p_id	付款 ID	INT	PK	Not Null, Unique	整數
or_id	訂單 ID	INT	FK: Order(or_id)	Not Null	整數
payment_method	付款方式	VARCHAR(10)		Not Null	字串 (支付方式)
payment_datetime	付款日期及時間	DATETIME		Not Null	日期時間格式
amount	付款金額	DECIMAL(10,2)		Not Null	正數
Referential triggers		On Delete	On Update		
or_id: Order(or_id)		Cascade	Cascade		

表 8: 資料表 PAYMENT 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
v_id	場地 ID	INT	PK	Not Null, Unique	整數
v_name	場地名稱	VARCHAR(15)		Not Null	字串 (名稱)
address	場地地址	TEXT			字串 (地址)
capacity	容量	INT		Not Null	正整數
contact_info	場地聯絡資訊	VARCHAR(15)			字串 (聯絡資訊)

表 9: 資料表 VENUE 的欄位資訊

Column Name	Meaning	Data Type	Key	Constraint	Domain
e_id	活動 ID	INT	FK: Event(e_id)	Not Null	整數
v_id	場地 ID	INT	FK: Venue(v_id)	Not Null	整數
arrangement	場地配置	TEXT			字串 (配置)
Referential triggers		On Delete	On Update		
e_id: Event(e_id)		Cascade	Cascade		
v_id: Venue(v_id)		Cascade	Cascade		

表 10: 資料表 EVENT_VENUE 的欄位資訊

2.4 正規化分析

以下是針對資料庫設計是否符合 1NF、2NF 和 3NF 的說明：

- **1NF** 要求所有欄位的值都不能是 multi-valued，即每個欄位的值必須是不可再分的單一值，且不能包含重複組合或子集合。在此資料字典中，每張表的欄位均為單一值，因此符合 1NF 的要求。
- **2NF** 要求所有非主鍵欄位完全依賴於主鍵，且不存在部分依賴。在此資料庫設計中，每張表的非主鍵欄位（例如 `EVENT` 表中的 `e_name` 和 `description`）皆完全依賴其主鍵（如 `e_id`），符合 2NF 的要求。
- **3NF** 要求每個非主鍵欄位必須直接依賴於主鍵，且不能依賴其他非主鍵欄位。在這個設計中，幾乎所有非主鍵欄位都直接依賴於主鍵，並且不存在傳遞依賴的情況，因此大部分符合 3NF 的要求。然而在 `TICKET` 表中，`t_id` 和 `e_id` 與 `t_type` 之間可能存在傳遞依賴 (transitively depends)。由於 Ticket 表主要紀錄票的狀態，而每個 `EVENT` 的 `t_type` 也不多，選擇不拆開也不太會影響資料的冗餘性，而拆解可能會降低資料庫的可讀性，因此選擇保留現有設計。

3 系統實作

3.1 資料庫建置方式及資料來源說明

關於 EVENT、TICKET 資料表內的資料建置方式，我們首先由活動主辦方提供活動資訊，包含活動名稱、日期、地點及簡介等，並由管理員在系統中新增這些活動資料。票券資訊則由管理員根據每場活動的需求發行不同類型的票券，設定票券價格、數量及其他相關屬性。

USER 資料表內的資料建置方式，採取使用者自行註冊的方式，使用者在註冊時需提供姓名、電子郵件及密碼。所有使用者註冊後的資料會被儲存至 USER 資料表中，並根據需求由管理員將特定使用者的角色設定為 Admin。

ORDER 與 ORDER_DETAIL 資料表則由使用者在購票過程中自動生成，當使用者完成購票後，系統會根據選擇的票券類型及數量，生成相應的訂單記錄並儲存至資料庫中。

為了要產生龐大的資料測試我們的資料庫，我們使用 python 中的 faker 函式庫來幫助我們產生隨機的資料，在產生資料的時候我們遵循著以下幾點以確保產生的資料是合理且貼近現實的資料。在生成資料之前我們預設我們生成的資料都是台灣地區常見的格式，首先看到 CATEGORY 這個資料表，由於表演種類分類明確且數量少，所以直接由我們指定資料表的內容並插入。再來從沒有 foreign key 的資料表開始隨機生成資料，例如 VENUE、ORGANIZER、CUSTOMER 等資料表，接著再從只需要依賴上面的表的資料表來生成資料，例如 EVENT、TICKET 以及 ORDER，最後才去生成剩下的資料表。所有資料表在隨機生成的時候都有符合應該要有的 PK、FK 關係。

除了隨機生成的資料外，我們也另外去網路上抓了一些實際的演唱會、運動賽事以及展覽的資料放進我們的資料庫，藉由隨機以及真實的混合資料增加資料庫內容的真實性，

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。

3.2.1 給 User 的功能

1. 註冊帳號：若要實現此功能，假設情境為「使用者想要註冊一個新帳號，用戶名稱為『user』，電子郵件為『user@example.com』，密碼為『password123』。」，系統會先檢查是否 Username 與電子郵件是否為唯一，若是，則會執行 SQL 指令，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。

```
INSERT INTO CUSTOMER (cu_name, pwd, email) VALUES ('user', '
```

```
        password123', 'user@example.com') RETURNING cu_id;
```

Listing 1: 註冊帳號 SQL 指令

2. 登入系統：若要實現此功能，假設情境為「使用者輸入用戶名稱『user』及密碼『password123』進行登入。」則對應的 SQL 指令如下。系統會在 USER 資料表中查找符合條件的使用者。

```
SELECT cu_id, role FROM CUSTOMER WHERE cu_name = 'user' AND
    pwd = 'password123';
```

Listing 2: 登入系統 SQL 指令

3. 瀏覽活動：若要實現此功能，假設情境為「使用者想要瀏覽體育類別的活動。」則對應的 SQL 指令如下。系統會在 EVENT 資料表中查詢目前有的所有類別，在根據使用者輸入的類別來找尋該類別的所有活動。

```
SELECT c_id, c_name FROM CATEGORY ORDER BY c_id;
```

Listing 3: 列出所有類別 SQL 指令

```
SELECT E.e_id, E.e_name, C.c_name, O.o_name, E.e_datetime, E
    .e_location
FROM EVENT E
JOIN CATEGORY C ON E.c_id = C.c_id
JOIN ORGANIZER O ON E.o_id = O.o_id
WHERE E.c_id = 8 -- sport 的 c_id
ORDER BY E.e_datetime;
```

Listing 4: 根據類別列出所有活動 SQL 指令

4. 尋找活動：若要實現此功能，假設情境為「使用者想要尋找『周杰倫』演唱會。」則對應的 SQL 指令如下。系統會在 EVENT 資料表中根據條件篩選活動或是對應的主辦方。

```
SELECT E.e_id, E.e_name, C.c_name, O.o_name, E.e_datetime, E
    .e_location
FROM EVENT E
JOIN CATEGORY C ON E.c_id = C.c_id
JOIN ORGANIZER O ON E.o_id = O.o_id
WHERE E.e_name ILIKE "周杰倫" OR O.o_name ILIKE "周杰倫"
```

```
    ORDER BY E.e_datetime;
```

Listing 5: 尋找活動 SQL 指令

5. 購買票券：若要實現此功能，假設情境為「使用者想要購買活動代號『101』的『VIP』票券，數量為『2』。」則對應的 SQL 指令如下。系統會在 TICKET 資料表中更新票券數量，並在 ORDER 與 ORDER_DETAIL 資料表中新增訂單紀錄。

```
-- 檢查票券是否有足夠數量
SELECT * FROM TICKET WHERE e_id = '101' AND t_type = 'VIP'
-- 更新票券數量
UPDATE TICKET SET remain_quantity = remain_quantity - '2'
   WHERE t_id = '1012'; --假設該票種 id 為 1012

-- 新增訂單
INSERT INTO "ORDER" (cu_id, or_date, total_amount,
   payment_status, is_canceled)
VALUES ('xx', current_date, 'xx', 'Pending', 'false')
   RETURNING or_id;

-- 創建訂單明細
INSERT INTO ORDER_DETAIL (or_id, t_id, quantity, subtotal)
VALUES ('xxx', '1012', 2, 'xxx');
```

Listing 6: 購買票券 SQL 指令

6. 取消訂單：若要實現此功能，假設情境為「使用者想要取消訂單代號『202』。」則對應的 SQL 指令如下。系統會在 ORDER 資料表中更新訂單狀態，並在 TICKET 資料表中恢復票券數量。

```
-- 檢查訂單是否可取消
SELECT payment_status, is_canceled FROM "ORDER" WHERE or_id
   = '202' AND cu_id = 'xxx';

-- 更新訂單狀態
UPDATE "ORDER" SET is_canceled = TRUE, payment_status = '
   Canceled' WHERE or_id = '202';
--恢復票券數量
SELECT t_id, quantity FROM ORDER_DETAIL WHERE or_id = '202';
```

```sql
UPDATE TICKET SET remain_quantity = remain_quantity + '2'
   WHERE t_id = '1012';
```

Listing 7: 取消訂單 SQL 指令

7. 查看活動詳情：若要實現此功能，假設情境為「使用者想要查看活動代號『101』的詳細資訊。」則對應的 SQL 指令如下。系統會在 EVENT 資料表中查詢特定活動的詳細資訊。

```sql
SELECT E.e_id, E.e_name, C.c_name, O.o_name, E.e_datetime, E
   .e_location, E.description
SELECT T.t_type, T.price, T.total_quantity
FROM EVENT E
JOIN CATEGORY C ON E.c_id = C.c_id
JOIN ORGANIZER O ON E.o_id = O.o_id
JOIN TICKET T ON T.e_id = E.e_id
WHERE E.e_id = 101;
```

Listing 8: 查看活動詳情 SQL 指令

8. 進行付款：若要實現此功能，假設情境為「使用者完成購票後，選擇信用卡付款，金額為『5000』，訂單 id 為 202。」則對應的 SQL 指令如下。系統會在 ORDER 資料表中更新付款狀態。

```sql
--檢查訂單是否存在且為待付款狀態
SELECT payment_status FROM "ORDER" WHERE or_id = '202';
--插入付款記錄
INSERT INTO PAYMENT (or_id, payment_method, payment_datetime
   , amount)
VALUES ('202', 'credit_card', current_datetime, '5000')
   RETURNING p_id;
--更新訂單狀態
UPDATE "ORDER" SET payment_status = 'Paid' WHERE or_id = '
   202';
```

Listing 9: 進行付款 SQL 指令

9. 查看購票歷史：若要實現此功能，假設情境為「使用者想要查看自己過去的所有訂單紀錄。」則對應的 SQL 指令如下。系統會在 ORDER 與 ORDER_DETAIL 資料表中查詢使用者的所有訂單紀錄。

```sql
SELECT O.or_id, E.e_name, T.t_type, OD.quantity, OD.subtotal
   , O.or_date, O.payment_status
FROM "ORDER" O
JOIN ORDER_DETAIL OD ON O.or_id = OD.or_id
JOIN TICKET T ON OD.t_id = T.t_id
JOIN EVENT E ON T.e_id = E.e_id
WHERE O.cu_id = 'user'
JOIN CUSTOMER C ON C.cu_id = O.cu_id
WHERE C.cu_name = 'user'
ORDER BY O.or_date DESC;
```

Listing 10: 查看購票歷史 SQL 指令

10. 編輯個人資訊：若要實現此功能，假設情境為「使用者想要更新自己的所有資訊。」則對應的 SQL 指令如下。系統會在 USER 資料表中更新使用者的資訊。

```sql
UPDATE CUSTOMER SET {field} = 'user' WHERE cu_id = 'xxx';
```

Listing 11: 編輯個人資訊 SQL 指令

3.2.2 給 Admin 的功能

1. 新增活動：若要實現此功能，假設情境為「管理員想要新增一場名為『《梵谷：尋光之路》特展』，日期與時間為『2024-08-24 09:00:00』，地點為『富邦美術館』，簡介為『梵谷誕生逾 170 週年! 台灣唯一站盛大展出中!』。」則對應的 SQL 指令如下。系統會在 EVENT 資料表新增一筆活動資料。

```sql
INSERT INTO EVENT (e_id, e_name, c_id, o_id, e_datetime,
   e_location, description)
VALUES (DEFAULT, "《梵谷：尋光之路》特展", 'xxx', 'xxx', "
   2024-08-24 09:00:00", "富邦美術館", "梵谷誕生逾 170 週年
   ! 台灣唯一站 盛大展出中!") RETURNING e_id;
```

Listing 12: 新增活動 SQL 指令

2. 發行票券：若要實現此功能，假設情境為「管理員想要為活動代號『101』發行『VIP』票券，價格為『3000』，數量為『100』。」則對應的 SQL 指令如下。系統會在 TICKET 資料表新增一筆票券資料。

```sql
-- 檢查活動是否存在
SELECT 1 FROM EVENT WHERE e_id = '101';
-- 檢查票種是否已存在
SELECT 1 FROM TICKET WHERE e_id = '101' AND t_type = 'VIP';
-- 插入票種
INSERT INTO TICKET (e_id, t_type, price, total_quantity,
   remain_quantity)
VALUES ('101', 'VIP', '3000', '100', '100') RETURNING t_id;
```

Listing 13: 發行票券 SQL 指令

3. 查詢使用者資訊：若要實現此功能，假設情境為「管理員想要查詢使用者代號『1』的詳細資訊。」則對應的 SQL 指令如下。系統會在 USER 資料表中查詢特定使用者的資訊。

```sql
SELECT cu_id, cu_name, email, phone_number, address, role
   FROM CUSTOMER WHERE cu_id = '1';
```

Listing 14: 查詢使用者資訊 SQL 指令

4. 查看使用者購票歷史：若要實現此功能，假設情境為「管理員想要查看使用者代號『1』的所有購票訂單紀錄。」則對應的 SQL 指令如下。系統會在 ORDER 與 ORDER_DETAIL 資料表中查詢使用者的所有訂單紀錄。

```sql
SELECT O.or_id, E.e_name, T.t_type, OD.quantity, OD.subtotal
   , O.or_date, O.payment_status
FROM "ORDER" O
JOIN ORDER_DETAIL OD ON O.or_id = OD.or_id
JOIN TICKET T ON OD.t_id = T.t_id
JOIN EVENT E ON T.e_id = E.e_id
WHERE O.cu_id = '1'
JOIN CUSTOMER C ON C.cu_id = O.cu_id
WHERE C.cu_name = 'user'
ORDER BY O.or_date DESC;
```

Listing 15: 查看使用者購票歷史 SQL 指令

5. 查看銷售報表：若要實現此功能，假設情境為「管理員想要查看 event_id 為『1』的銷售報告。」則對應的 SQL 指令如下。

```sql
SELECT
```

```
    T.t_type,
    SUM(OD.quantity) AS tickets_sold,
    SUM(OD.subtotal) AS revenue
FROM TICKET T
JOIN ORDER_DETAIL OD ON T.t_id = OD.t_id
JOIN "ORDER" O ON OD.or_id = O.or_id
WHERE T.e_id = '1' AND O.payment_status = 'Paid'
GROUP BY T.t_type;
```

Listing 16: 查看使用者購票歷史 SQL 指令

3.3 SQL 指令效能優化與索引建立分析

3.3.1 在 ORDER 中加入使用者代號與訂單日期的索引

我們認為購票系統中最常使用到的功能之一為「查看購票歷史」，因此我們決定針對該指令進行效能優化。此功能需要根據使用者代號來查詢訂單紀錄。由於 ORDER 資料表中的資料量龐大，為了加速查詢速度，我們針對 User_id 及 Order_date 欄位建立了複合索引。

為了優化效能，我們在 ORDER 資料表中建立了如下的索引：

```
CREATE INDEX idx_order_cu_id_or_date
ON "ORDER" (cu_id, or_date);
```

Listing 17: 建立 ORDER 表的 index

透過這個索引，系統在執行「查看購票歷史」功能時，可以更快速地定位到特定使用者的訂單紀錄，顯著提升查詢效率。

3.3.2 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「查看購票歷史」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 68 毫秒、58 毫秒、60 毫秒、69 毫秒、83 毫秒，平均運行時間約為 67.6 毫秒，標準差約為 8.822 毫秒；建立索引後運行五次的結果分別是 56 毫秒、57 毫秒、56 毫秒、58 毫秒、61 毫秒，平均運行時間約為 57.6 毫秒，標準差約為 1.854 毫秒。

透過這些測試結果，我們可以看到索引的建立對於查詢效能有著顯著的提升，特別是在資料量龐大的情況下，索引的作用更加明顯。

3.4 交易管理

在購票過程中，涉及多個資料表的操作，如購買票券時需要同時更新 TICKET、ORDER 及 ORDER_DETAIL 等資料表。為了確保資料的一致性與完整性，我們採用了交易管理的方式，確保所有操作只能全部成功或全部失敗，避免部分操作成功導致資料不一致。

例如，當使用者購買票券時，系統會啟動一個交易，先檢查票券是否有足夠數量，若有，則扣除票券數量並新增訂單及訂單明細。若在任一步驟中出現錯誤，系統會 Rollback 交易，恢復所有資料表的狀態，確保資料的一致性。

3.5 併行控制

在購票系統中，可能有多位使用者同時嘗試購買同一類型的票券，導致票券數量的競爭性修改。為了避免超賣現象，我們採用了併行控制機制，確保在同一時間內只有一個交易能夠修改票券數量。

我們使用資料庫的鎖機制（如行鎖）來保護 TICKET 資料表中的票券數量欄位。當一個交易正在修改某票券的數量時，其他交易必須等待該交易完成後才能進行操作，從而避免同時修改導致的數量錯誤。

例如，當使用者 A 嘗試購買票券時，系統會對該票券的行進行鎖定，直到交易完成或回滾，其他使用者 B、C 等在此期間的購票請求將會被擋住，等到鎖釋放後再進行處理。

4 分工資訊

本專案由第 11 組組員共同完成，組員分工如下：

- 江彥宏：系統分析、資料處理。
- 陳奕廷：前端開發，實作使用者介面與互動功能。
- 廖振翔：系統設計，資料庫架構設計，後端開發，與資料庫互動。

5 專案心得

- 江彥宏：這次期末專案剛好和很多科都擠在一起，加上要做的事還不少，因此這份專案也是在很趕的時間內做出來的，光是要把資料生出來就要花了一番工夫，畢竟資料必須符合常理、且符合資料庫的設計。而且這次的期末專案分工幾乎是三個人做著各自的工作，有人負責處理資料、有人負責前端、有人負責後端，當大家把自己的工作做好之後彼此要互相結合的時候也是一種有趣的地方。

- 陳奕廷：這次的專案實在是要做的事情有點多，而且老師限定一組最多三個人，因此每個人對專案的貢獻都要很高，我們組員也都很認真的完成了。這次我負責的是前端，我製作了一個還算可以用的網頁，雖然沒考慮到的東西還是很多，不過也算是讓我初次成功串接起一個前端跟後端。前端後端的串聯常常出現資料接不到，後端連不到的問題，也是挺惱人的。這次也學到很多資料庫上課所說的一些 Lock 以及 DBMS 自己的防呆對專案來說有多重要，還有包含前半學期所學的 Schema，沒想到我們可以把資料庫建立起來，這讓我覺得這門課沒有白學，挺有成就感的。

- 廖振翔：這次的專案真的要做太多事情，而且只有三個人，還要從零開始架設一個系統，包括資料庫設計，系統架構，前後端，與資料庫的連接，生成/找資料，在做這個系統中我覺得最難兼顧到的就是整個系統的可擴充性和完整性，因為這可以算是我第一次從零開始做出一個算堪用的系統，很高興可以參照助教範例專案的整體架構，給了我一開始的大方向，不然可能會寫出一個幾千行的 server 跟 client 全部塞在一起了。