114-1 資料庫管理 期末專案完整報告

第十組

資管三 B11705061 羅立宸
資管三 B12705011 黃元翔
資管三 B12705057 陳以倫

Dec. 2025

GitHub 專案連結 YouTube 影片連結

1 系統分析

學期初買的教科書超貴，學期末卻堆在角落長灰塵？想買學長姐的二手書，卻不知道去哪裡找？如果你有這些困擾，趕緊上「BookSwap」尋找你需要的二手好物！

「BookSwap」是一個提供給某大學學生刊登及尋找二手教科書與物品的平台，主要目的是幫助該校學生解決二手物品（特別是教科書）資訊不流通、交易媒合困難的窘境。平台上的「刊登」是指一次性的物品出售貼文，每篇貼文都是獨立的，有自己的刊登編號。交易完成或下架後，該刊登即失效。使用者可以主動舉報可疑貼文或留言，建立社群自我管理的基礎。每次違規都會被記錄，達一定次數（例如 3 次）後，系統會自動停權或封鎖帳號，防止惡意刊登或詐騙帳號重複出現。

根據不同的功能及掌控權限，「BookSwap」系統的用戶可以分為兩種身分，分別是 User 及 Admin 。User (一般使用者)：可依照自身需求，選擇「刊登」想賣的物品，或是「瀏覽/搜尋」平台上由其他人刊登的物品。若想刊登物品，可透過介面輸入物品的標題、描述、價格、分類，並上傳照片。如果物品是教科書，還可以關聯到特定「課程」。如果使用者想購買物品，則可瀏覽平台上現有的刊登，並選擇感興趣的物品進行聯繫。Admin（管理者）：則是「BookSwap」系統的業務經營者，主要負責管理「課程」及「物品分類」的資訊，並且可查詢所有使用者的刊登紀錄，收到 User 舉報後會審核、決定是否移除或警告不適當的刊登內容。

1.1 系統功能

1.1.1 關於刊登的相關設定

系統會提供「物品分類」讓使用者選擇，例如：教科書、3C 產品、生活用品等。如果使用者選擇的分類是「教科書」，系統會建議使用者從一個課程列表中選擇想關聯的「課程」，例如「資料庫管理」，以利他人搜尋。每則刊登都會有「狀態」，例如：刊登中、預訂中、已售出。

1.1.2 給 User 的功能

在本系統中，User 可以執行以下功能：

- 新增刊登：使用者能透過設定物品標題、描述、價格、分類等相關資訊來刊登一項物品。如果物品是教科書，還可以選擇想關聯的課程。一旦刊登，系統便會給定一個屬於該刊登的編號。
- 新增留言：使用者能透過在刊登的底下留言，例如" 想要"、" 有興趣" 來表達自己的意願，也能使刊登者知道自己刊登的物品是否有人有意願。
- 收藏物品：使用者若看到感興趣的刊登，可將其「加入收藏」，作為一筆新的收藏資料新增至資料庫。
- 管理刊登：使用者如果不想繼續販售，可刪除（或下架）自己刊登的物品。
- 查詢使用者曾經刊登過的物品：使用者可以查詢自身曾建立過的刊登和刊登的所有留言，包括「刊登中」與「已售出」的。
- 查詢使用者收藏的物品：使用者可以查詢自身收藏過的刊登。
- 查詢目前平台上的刊登：使用者可依分類、課程、關鍵字或價格，查詢尚未售出且刊登中的物品。
- 舉報不當刊登：若使用者發現疑似詐騙、違禁品或不當內容的貼文，可透過「舉報」功能提交檢舉。

1.1.3 給 Admin 的功能

在本系統中，Admin 可以執行以下功能：

- 管理課程：業務經營者可對課程資訊進行增刪改查的操作，以確保課程列表是最新狀態。
- 管理分類：業務經營者可對物品分類進行增刪改查的操作。
- 查詢使用者資訊：業務經營者可查詢所有使用者的活動紀錄，包括該使用者曾經刊登過哪些物品。
- 查詢刊登資訊：業務經營者可查詢所有刊登的詳細資訊，並移除違規（如詐騙或違禁品）的刊登。
- 審核舉報與處理違規：業務經營者會審核舉報內容，若經查屬實，將移除該刊登並記錄違規行為。當使用者違規達一定次數後，系統會自動封鎖該使用者帳號，以維護平台秩序與安全。

2 系統設計

2.1 ER Diagram

Figure 1 是 BookSwap 的 ER Diagram，這個 ERD 中有三個實體：User, Posting, Comment 以及三個關係：Post, Make_Comments, Have。

其中 User 代表的是使用「BookSwap」的使用者，我們讓使用者用 email 註冊，並請使用者設定一組 username 和 password，在註冊後我們會給使用者一

個獨一無二的 ID，並且可以在使用者創建帳戶後，從後台將其變更為管理者，此時該使用者的 Admin 就會是 1，否則為 0。

每位使用者都可以將一些他喜歡的 post 放入自己的 favorites 清單。另外如果使用者的刊登被下架超過三次，這位使用者也會被封鎖。

Posting 代表的是使用者刊登的物品，每個物品都有一個獨一無二的 ID，會有標題、說明文字、價格、狀態（刊登中、已售出、已被檢舉、已下架）、關聯的課程（如果物品是教科書）、課程所屬的類別（Class）比如說法律類、資訊類、文學類。

Comment 代表是每個 Posting 實體下的留言，每則留言都會有獨特的 ID、內容、由哪位使用者發出。

Post 關係是指一位使用者可以刊登多筆刊登。Make_Comments 關係是指一位使用者可以留多筆言。Have 關係是指一筆刊登可以擁有多筆留言。

ERD 中可能有部分不符合 normalization 的設計，這些會在 2.2 節解決。

2.2 Relational Database Schema Diagram

Relational Database Schema Diagram 中總共有 7 個 table，分別是 `user` (使用者)、`posting` (貼文)、`comment` (留言)、`class` (貼文類別)、`course` (課程)、`favorite_posts` (收藏貼文) 和 `report` (舉報)。如下圖 Figure 2 所示。

2.3 Data Dictionary

資料表共有 Relational Database Schema Diagram 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
u_id	使用者編號	INT UNSIGNED	PK	Not Null, Auto_Increment	正整數流水號
is_admin	是否為管理者	TINYINT(1)		Not Null, Default 0	{0：一般使用者，1：管理者}
user_name	使用者名稱	VARCHAR(50)		Not Null, Unique	
email	電子郵件	VARCHAR(100)		Not Null, Unique	符合 Email 格式
password	密碼	VARCHAR(8)		Not Null	登入用密碼
status	使用者帳號狀態	ENUM(...)		Not Null, Default 'normal'	{'normal','banned'}
violation_cnt	違規次數	INT UNSIGNED		Not Null, Default 0	達一定次數自動停權
balance	錢包餘額	INT		Not Null, Default 0	單位：元，新台幣

Table 1: 資料表 USER 的欄位資訊

2.4 正規化分析

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 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），且此關聯滿足 1NF，則滿足 2NF。

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。因此同樣檢視一下設計的關聯，的確有符合 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），也就是要確保 X → Y 的 X 一定是超級鍵。我們的 schema 也符合 BCNF。

最後是 4NF，由於「BookSwap」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

本系統為確保功能測試之完整性與展示效果，採用自動化腳本批量生成模擬資料（Mock Data）。資料建置並非採用手動輸入，而是透過編寫 PostgreSQL 的 PL/pgSQL 程序化語言腳本（generate/_fake/_data.sql），依據預設邏輯自動填充資料庫。具體的資料建置策略與來源說明如下：

1. 資料生成策略 (Data Generation Strategy)

自動化生成：利用 SQL 迴圈（Loop）與隨機函數，動態產生使用者資訊、商品詳情與交易紀錄，模擬系統長期運作下的數據累積。

真實性模擬：基礎資料：預先匯入真實存在的 20 個科系（如資管、資工、法律等）與 20 門常見課程（如資料庫管理、經濟學原理），確保學術背景的真實感。內容差異化：針對不同分類（如教科書、3C 產品）設定差異化的價格區間與標題格式，並串接外部圖片服務（Placeholder API）模擬商品圖片。

2. 邏輯完整性與關聯控制 (Logic & Integrity Constraints)

為確保資料符合資料庫正規化原則與業務邏輯，生成過程嚴格遵守以下限制：

參照完整性 (Referential Integrity)：嚴格依照「基礎資料 → 使用者 → 商品刊登 → 互動/交易」的順序生成，確保所有外鍵關聯（Foreign Key）正確無誤。

業務邏輯檢查：買賣邏輯：系統強制限制買家與賣家不得為同一使用者。狀態連動：僅有狀態為「已售出 (Sold)」的商品才會生成對應訂單；僅有「已完成 (Completed)」的訂單才會產生評價。價格一致性：訂單成交價格嚴格對應商品刊登時的原始價格。

3. 資料規模 (Data Scale)

目前測試環境共生成約 60,000 筆資料，涵蓋系統核心的 13 張資料表，規模足以進行壓力測試與查詢效能分析：使用者：1,000 位（包含雜湊處理過的密碼與隨機餘額）。商品刊登：12,000 筆（涵蓋上架、保留、售出等不同狀態）。交易與互動：包含 3,000 筆訂單、5,000 筆交易紀錄及數千筆留言與收藏紀錄。

3.2 重要功能及對應的 SQL 指令

在第 1.1 節中，我們介紹了 BookSwap 平台提供給一般使用者（User）與管理員（Admin）的主要功能，以及系統在背景中自動執行的一些維護操作。本小節將以具體情境為例，說明這些功能在實作上對應到的 SQL 指令或資料庫函數呼叫。第 3.2.1 小節聚焦於給一般使用者使用的功能，第 3.2.2 小節則說明管理員相關的查詢與維護操作，最後第 3.2.3 小節則整理由系統本身運行的「系統級指令」。

3.2.1 給 User 的功能

1. **刊登二手書：新增一筆商品刊登**
   若要實現「刊登二手書」功能，假設情境為：「使用者代號 `u_id` 為 10 的學生，想刊登一本標題為『資料庫管理（第 3 版)』的教科書，售價 `price` 為 450 元，並指定分類 `class_id` 為 2、對應課程 `course_id` 為 15。」

對應的 SQL 指令如下。系統會在 posting 資料表新增一筆刊登紀錄，初始狀態 status 設為 'listed' 代表可供購買。

```
INSERT INTO posting (u_id, title, description, price, status,
                     class_id, course_id)
VALUES (10,
        '資料庫管理（第 3 版）',
        '九成新，書內有少量筆記，可面交',
        450,
        'listed',
        2,
        15)
RETURNING p_id;
```

透過 PostgreSQL 的 RETURNING 子句，系統可以立即取得新建立的 p_id，後續若需要再為該刊登新增多張圖片，便能將此編號插入 posting_images 資料表中。

```
INSERT INTO posting_images (p_id, image_url, display_order)
VALUES (/* 上一步取得的 p_id */,
        'https://example.com/images/book1.jpg',
        1);
```

2. **搜尋可購買的書籍：依課程名稱與價格篩選刊登**
使用者在前端可以輸入關鍵字、選擇課程或分類，來查詢目前可購買的刊登。假設情境為：「使用者想查詢課程名稱內含『資料庫』，並且價格介於 0 到 500 元之間，且商品狀態為可購買（listed）的刊登。」
對應的 SQL 指令如下。系統會回傳符合條件的刊登資訊，包含課程名稱、分類名稱與賣家帳號：

```
SELECT p.p_id,
       p.title,
       p.price,
       p.status,
       u.username      AS seller_username,
       co.course_name,
       cl.class_name
FROM posting AS p
JOIN "user" AS u   ON p.u_id = u.u_id
LEFT JOIN course AS co ON p.course_id = co.course_id
LEFT JOIN class  AS cl ON p.class_id = cl.class_id
WHERE p.status = 'listed'
  AND co.course_name ILIKE '%資料庫%'
  AND p.price BETWEEN 0 AND 500
ORDER BY p.created_at DESC;
```

此查詢會利用我們在 posting(status)、posting(price)、course(course_id) 等欄位上建立的索引，降低掃描資料表的成本。

若使用者改以關鍵字搜尋書名 / 描述，我們則會使用 PostgreSQL 全文搜尋，對應到資料庫中以 GIN 索引加速的 to_tsvector 欄位：

```
SELECT p.p_id, p.title, p.price, p.status
FROM posting AS p
WHERE p.status = 'listed'
  AND to_tsvector('english', p.title || ' ' || COALESCE(p.description, ''))
      @@ plainto_tsquery('english', 'database');
```

3. **將刊登加入收藏清單：新增收藏紀錄**
   若要實現「加入收藏」功能，假設情境為：「使用者代號 u_id 為 10，想把刊登編號 p_id 為 123 的書籍加入收藏。」
   對應的 SQL 指令如下。系統會在 favorite_posts 資料表中新增一筆紀錄，若日後再次查詢收藏清單，即可快速找到這筆刊登。

   ```
   INSERT INTO favorite_posts (u_id, p_id)
   VALUES (10, 123);
   ```

4. **購買書籍：呼叫交易函數 purchase_book**
   當使用者決定購買某本書時，前端會將買家代號與刊登編號送往後端，由後端透過 PostgreSQL 的 PL/pgSQL 函數 purchase_book 執行完整的交易流程。假設情境為：「使用者 u_id = 10 想購買刊登 p_id = 123 的書籍。」
   對應的 SQL 呼叫如下：

   ```
   SELECT purchase_book(10, 123);
   ```

   此函數內部會依序檢查刊登是否存在、狀態是否為 'listed'、買家餘額是否足夠、是否為自己購買自己的商品，並在同一個交易中同步完成扣款、入帳、更新刊登狀態與建立訂單，確保購買流程具有 ACID 特性。

5. **在刊登底下留言：新增公開留言**
   為了讓買家可以在刊登底下詢問細節（例如是否可面交、是否有畫線），我們提供「公開留言」功能。假設情境為：「使用者 u_id = 10 想在刊登 p_id = 123 底下留言『請問可以在總圖面交嗎？』。」
   對應的 SQL 指令如下：

   ```
   INSERT INTO comment (p_id, u_id, content)
   VALUES (123, 10, '請問可以在總圖面交嗎？');
   ```

   系統會自動記錄留言時間 created_at，後續在刊登頁面讀取留言時，會依照時間排序顯示。

6. **傳送私訊：建立一則一對一訊息**
   若買家與賣家希望進一步溝通（例如交換 Line、約定面交時間），系統會透過 message 資料表記錄雙方的一對一訊息。假設情境為：「使用者 u_id = 10 想私訊刊登的賣家 u_id = 25。」
   對應的 SQL 指令如下：

   ```
   INSERT INTO message (sender_id, receiver_id, content)
   VALUES (10, 25, '您好，請問這本書還在嗎？可以約下週一面交嗎？');
   ```

   當訊息送出後，前端會根據 is_read 欄位顯示未讀提示，並透過索引 idx_message_receiver_id 與 idx_message_is_read 加速收件匣查詢。

3.2.2 給 Admin 的功能

1. **管理課程與分類（增刪改查）**
   管理員可以維護系上課程、系所與分類資訊，讓刊登可以正確對應到實際課程。假設情境為：「管理員想新增一門課程『資料庫管理』，屬於『資訊管理學系』且歸類在『必修』這個分類中。」
   對應的 SQL 指令如下。首先新增系所與分類（若尚未存在），再新增課程：

```
INSERT INTO department (dept_name)
VALUES ('資訊管理學系');

INSERT INTO class (class_name, description)
VALUES ('必修', '系上必修課程');

INSERT INTO course (course_code, course_name, dept_id, class_id)
VALUES ('IM3001', '資料庫管理', 1, 1);
```

   若日後需要修改課程代碼或名稱，管理員可以使用 UPDATE 指令：

```
UPDATE course
SET course_name = '資料庫管理（含實作）'
WHERE course_id = 1;
```

2. **處理舉報：審核留言 / 刊登 / 逃單舉報**
   當使用者發現不當內容或交易糾紛時，可以透過 report 資料表送出舉報。管理員在後台審核時，會將狀態從 'pending' 更新為 'approved' 或 'rejected'。假設情境為：「管理員 u_id = 1 審核編號為 50 的舉報，決定通過此舉報。」
   對應的 SQL 指令如下：

```
UPDATE report
SET status      = 'approved',
    reviewed_by = 1,
    reviewed_at = CURRENT_TIMESTAMP
WHERE report_id = 50;
```

   當舉報被標記為 'approved' 時，觸發器 update_violation_count 會根據舉報類型（刊登、留言或逃單）自動找到對應的目標使用者，並將其 violation_count 加一；當違規次數累積到三次，另一個觸發器 auto_block_user 會自動將該帳號的 is_blocked 欄位設為 TRUE，達到半自動的風紀管理效果。

3. **查詢使用者整體表現與風險狀態**
   為了讓管理員能快速掌握每位使用者在平台上的活動概況，我們在資料庫中建立視圖 v_user_statistics，整理了刊登數量、售出數量、購買金額、總收入、平均評分與收藏數等指標。若管理員想查詢 u_id = 10 的統計資訊，對應的 SQL 指令如下：

```
SELECT *
FROM v_user_statistics
WHERE u_id = 10;
```

此視圖將 "user"、posting、orders、review 與 favorite_posts 等多個資料表的資訊整合在一起，管理員只需一個查詢即可檢視使用者是否為高風險對象（例如違規次數偏高、評價過低或逃單紀錄較多）。

4. **查詢熱門書籍與課程統計**
   為了協助平台調整推薦策略與營運方向，我們另外建立了 v_popular_books、v_course_statistics、v_class_statistics 等視圖。若管理員想查詢目前收藏數較多的熱門書籍，對應的 SQL 指令如下：

```
SELECT *
FROM v_popular_books
WHERE favorite_count >= 5
ORDER BY favorite_count DESC, comment_count DESC;
```

若要觀察特定課程（例如 IM3001 資料庫管理）的整體交易狀況，則可使用下列查詢：

```
SELECT *
FROM v_course_statistics
WHERE course_code = 'IM3001';
```

藉由這些彙總視圖，管理員可以了解哪些課程或分類上的二手書需求特別高，作為未來功能優化與行銷活動的依據。

3.2.3 系統級指令

除了由使用者或管理員直接觸發的操作外，BookSwap 也在資料庫端設計了多個「系統級」的 SQL 函數與觸發器，用來在背景中維護金流紀錄與帳號風紀，避免出現資料不一致的情況。

1. **計算使用者平均評分：calculate_user_rating**
   為了在使用者頁面顯示賣家的整體評價，我們在資料庫中實作了 calculate_user_rating(p_user_id INT) 函數，用來計算指定使用者作為被評價者（target_id）時，其所有評分的平均值。假設系統需要取得 u_id = 10 的平均評分，對應的 SQL 呼叫如下：

```
SELECT calculate_user_rating(10);
```

此函數會忽略尚未有評分的情況（回傳 0），並將結果四捨五入至小數點後兩位，方便直接顯示在前端介面上。

2. **取得使用者銷售統計：get_user_sales_stats**
   另一個系統級函數 get_user_sales_stats(p_user_id INT) 則會回傳一個 JSON，其中包含使用者售出的書籍數量、總收入、目前上架中的刊登數量，以及平均評分等資訊。假設系統想取得 u_id = 10 的銷售統計，對應的 SQL 呼叫如下：

```
SELECT get_user_sales_stats(10);
```

這個函數主要用於後台儀表板與使用者個人頁面，讓前端可以一次取得多個統計欄位，減少與資料庫往返的次數。

3. **訂單完成時自動更新刊登狀態與金流紀錄**
   在第 3.4 節中我們已詳細說明購買流程的交易管理。這裡補充說明與之對應的系統級指令：當 `orders` 資料表中新增或更新一筆訂單，且其狀態為 `'completed'` 時，觸發器 `update_posting_status_on_order` 與 `record_transaction_on_order` 會自動執行下列 SQL 片段：

```
-- 1. 將對應的刊登狀態改為 sold
UPDATE posting
SET status    = 'sold',
    updated_at = CURRENT_TIMESTAMP
WHERE p_id = NEW.p_id AND status = 'listed';

-- 2. 在 transaction_record 中新增兩筆金流紀錄
INSERT INTO transaction_record (u_id, amount, trans_type)
VALUES (NEW.buyer_id, -NEW.deal_price, 'payment');  -- 買家付款

INSERT INTO transaction_record (u_id, amount, trans_type)
VALUES (seller_id, NEW.deal_price, 'income');       -- 賣家收入
```

如此一來，即使應用程式端只需插入或更新一筆訂單紀錄，資料庫也能自動確保刊登狀態與金流紀錄保持一致，避免出現「訂單完成但金流未記錄」或「金流有紀錄但刊登仍顯示可購買」等不一致情況。

3.3 SQL 指令效能優化與索引建立分析

3.3.1 User 表索引

我們觀察到系統中最頻繁的操作之一是與使用者相關的查詢，例如登入驗證、判斷帳號是否為管理員、篩選遭封鎖的用戶等。由於 email 與 username 是辨識使用者身分的重要欄位，且在註冊、登入與權限驗證流程中會被大量使用，因此若每次查詢都需在資料表逐筆比對，勢必造成效能負擔。

此外，is_admin 與 is_blocked 這兩個欄位經常被用於篩選條件，例如後台管理需要快速定位管理員帳號，或系統需要查詢遭封禁的使用者狀態。若無索引輔助，資料庫必須進行全表掃描 (Full Table Scan)，在使用者數量成長後，將會造成明顯延遲。

因此，為提升查詢效率，我們在 user 表中針對 email、username、is_admin 與 is_blocked 建立索引，以加快系統在使用者查詢行為中的回應速度。其語法如下。

```
1 CREATE INDEX IF NOT EXISTS idx_user_email
2 ON "user"(email);
3
4 CREATE INDEX IF NOT EXISTS idx_user_username
5 ON "user"(username);
6
7 CREATE INDEX IF NOT EXISTS idx_user_is_admin
8 ON "user"(is_admin);
9
```

```
CREATE INDEX IF NOT EXISTS idx_user_is_blocked
ON "user"(is_blocked);
```

上述索引的建立，使常見查詢如「搜尋 E-mail 是否已註冊」、「驗證使用者名稱是否合法」、「取得所有管理員帳號」與「查詢封鎖用戶」能夠避免全表掃描，提升資料檢索效率，並強化系統的使用體驗與可擴展性。

3.3.2 Posting 表索引

在系統的運作流程中，posting 資料表扮演核心角色，包含貼文內容、分類、價格、課程代碼與貼文建立時間等資訊。由於平台上的貼文瀏覽、篩選與搜尋操作頻繁，若每次查詢皆需逐筆比對，將造成系統效能下降。因此，我們決定針對常用查詢欄位建立索引，以提高查詢速度。

其中，u_id 會用於取得使用者的所有貼文；status 會頻繁用於顯示有效貼文或過期貼文的篩選；class_id 與 course_id 則有助於快速查詢特定課程或分類下的貼文內容。price 與 created_at 亦為排序及篩選熱門或最新貼文時不可或缺的條件，因此透過索引，我們可以有效降低查詢時間。

除了以上欄位索引外，我們亦為 title 與 description 建立全文搜尋 (Full Text Search) 索引，使用 PostgreSQL GIN + Tsvector 技術，以提升關鍵字搜尋效率與回傳精準度，使使用者能夠快速找到符合需求的貼文內容。

其語法如下。

```
-- Posting 表索引
CREATE INDEX IF NOT EXISTS idx_posting_u_id ON posting(u_id);
CREATE INDEX IF NOT EXISTS idx_posting_status ON posting(status);
CREATE INDEX IF NOT EXISTS idx_posting_class_id ON posting(class_id);
CREATE INDEX IF NOT EXISTS idx_posting_course_id ON posting(course_id);
CREATE INDEX IF NOT EXISTS idx_posting_price ON posting(price);
CREATE INDEX IF NOT EXISTS idx_posting_created_at ON posting(created_at
    );

-- 全文搜尋索引 (PostgreSQL)
CREATE INDEX IF NOT EXISTS idx_posting_title_search
ON posting USING gin(to_tsvector('english', title));

CREATE INDEX IF NOT EXISTS idx_posting_description_search
ON posting USING gin(to_tsvector('english', description));
```

透過此索引設計，系統能夠在貼文數量增加時仍維持快速回應，特別是篩選搜尋、課程查詢、價格排序與全文關鍵字查詢皆大幅提升效能，有助於貼文瀏覽與交易流程更加順暢。

3.3.3 Comment 表索引

在互動功能中，留言系統是使用者參與貼文內容的重要環節，comment 資料表則用於儲存各篇貼文的留言紀錄。為了確保系統能夠即時顯示留言、查詢特定貼文相關的討論內容，我們針對常用的查詢條件進行索引設計。

其中，p_id (貼文編號) 是最常用來查詢留言的條件，系統在顯示貼文下方留言時會頻繁依據此欄位提取資料，因此建立索引能有效降低 Full Table Scan 的成本。同時，u_id 可加速查詢某位使用者留下的所有留言，例如顯示個人留言記錄或追蹤違規帳號留言行為。至於 created_at 則常作為排序依據（最新留言在前），亦有助於以時間維度讀取留言串的效能。

其語法如下。

```
CREATE INDEX IF NOT EXISTS idx_comment_p_id ON comment(p_id);
```

```
CREATE INDEX IF NOT EXISTS idx_comment_u_id ON comment(u_id);
CREATE INDEX IF NOT EXISTS idx_comment_created_at ON comment(created_at
    );
```

透過此索引設置，系統在讀取留言串、顯示討論內容、回朔使用者留言紀錄時皆能更快速回應，並確保在留言數增加的情況下仍具備高擴充性與良好使用體驗。

3.3.4 Report 表索引

在檢舉與審核流程中，report 資料表負責儲存使用者針對貼文所提報的違規紀錄，因此查詢效率直接影響到審核速度與管理端的使用體驗。由於平台在運作過程中，管理者會大量針對檢舉紀錄進行查詢、篩選與排序，我們在設計上對常用查詢欄位建立索引，以強化系統在多筆檢舉情況下的效能。

其中，reporter_id 主要用於追蹤同一位使用者提出的所有檢舉行為，可協助判斷是否存在濫用申訴或重複檢舉。p_id 則對應貼文本身，可加速取得某則貼文涉及的所有檢舉事件，特別是熱門貼文被大量舉報時更能有效減少查詢負擔。status 為審查狀態，建立索引後可快速篩選未處理、通過或駁回的案件，大幅提升後台審核流程效率。而 created_at 則讓系統能依時間排序與取得最新檢舉紀錄，對時序分析與審核排序十分關鍵。

```
CREATE INDEX IF NOT EXISTS idx_report_reporter_id ON report(reporter_id
    );
CREATE INDEX IF NOT EXISTS idx_report_p_id ON report(p_id);
CREATE INDEX IF NOT EXISTS idx_report_status ON report(status);
CREATE INDEX IF NOT EXISTS idx_report_created_at ON report(created_at);
```

透過上述索引設計，後台管理者能更快速定位檢舉來源、查詢涉及問題的貼文、掌握處理進度並依時間排序案件，使審核流程具備更高反應速度與可處理上限，在資料量增加時亦保持良好延展性。

3.3.5 Orders 表索引

在交易流程設計中，orders 資料表負責紀錄使用者之間的交易訂單資訊，包含購買者、對應貼文、訂單狀態與下單時間等欄位。由於訂單查詢是整體系統運作的重要核心，例如買家查看訂單紀錄、賣家確認商品是否售出、後台篩選訂單狀態等場景都會大量依賴此資料表，因此若無索引輔助，在訂單數量增加後將出現顯著查詢延遲。

其中，buyer_id 用於取得某位使用者的購買紀錄，使平台能快速顯示歷史訂單、評價來源或交易分析結果。p_id 則對應 posting，可快速確認某篇貼文所產生的所有訂單，有助於追蹤熱門貼文的交易量。status 常用於後台審核與訂單處理，如顯示已完成、待付款或已取消的訂單狀態，索引能使篩選條件查詢更流暢。最後，order_date 使系統能快速依時間排序訂單，在報表生成、營運分析與近期交易查詢中具有重要作用。

索引建立語法如下。

```
CREATE INDEX IF NOT EXISTS idx_orders_buyer_id ON orders(buyer_id);
CREATE INDEX IF NOT EXISTS idx_orders_p_id ON orders(p_id);
CREATE INDEX IF NOT EXISTS idx_orders_status ON orders(status);
CREATE INDEX IF NOT EXISTS idx_orders_order_date ON orders(order_date);
```

透過以上索引配置，系統能在訂單量成長後仍保持查詢效率，包含查詢購買紀錄、對應貼文之訂單、依狀態分類訂單處理、以及依時間排序交易資料，都能顯著降低查詢延遲，確保交易流程與後台統計分析運作順暢。

3.3.6 Transaction Record 表索引

在平台金流與點數制度中，transaction_record 表用於儲存使用者的錢包交易紀錄，包含轉入、扣款、退款、購買等不同類型的金流行為。由於此資料與訂單、貼文交易、錢包餘額顯示等功能密切相關，交易紀錄的快速查詢對整體使用體驗與後台審計都具有關鍵影響。因此，我們針對常用查詢條件建立索引，以確保金流紀錄在長期使用下仍能維持高效運作。

其中，u_id 是最常用的查詢欄位，用於取得某位使用者的所有錢包紀錄，例如顯示充值紀錄、消費歷史或違規退款事件；建立索引後，可顯著加速此類查詢需求。trans_type 則用於類型識別，方便後台統計特定交易類型的發生次數，如統計每日扣款量、每日充值量等，索引可提升分類查詢的效率。trans_time 則非常適合用於時間排序與報表分析，當需取得近期交易或回朔歷史紀錄時，有索引的情況下可避免進行全表掃描，使查詢更具延展性。

建立索引之 SQL 如下所示：

```
CREATE INDEX IF NOT EXISTS idx_transaction_u_id ON transaction_record(
    u_id);
CREATE INDEX IF NOT EXISTS idx_transaction_trans_type ON
    transaction_record(trans_type);
CREATE INDEX IF NOT EXISTS idx_transaction_trans_time ON
    transaction_record(trans_time);
```

透過以上索引設計，交易紀錄在查詢使用者歷史、類型分類統計、依時間排序時皆具備更佳查詢效率，並能在資料量累積下保持後台分析、錢包查詢與系統審計的即時性與穩定度。

3.3.7 Review 表索引

評價系統是平台信任機制的重要組成，review 資料表負責儲存訂單完成後的評分與評論內容，用於反映交易品質、使用者行為、以及是否具備良好風評。由於評價資料經常用於排序、查詢、身份追溯、後台稽核等功能，因此若缺乏索引，當評論量逐漸成長後查詢將明顯變慢。因此，我們針對高頻查詢欄位建立索引以提升系統效能。

其中，order_id 可快速定位某筆交易是否已有評價，以及查看買賣雙方對單一訂單的回饋；reviewer_id 則用於查看某位使用者發表的所有評價，有助於分析買家或賣家在平台上的使用紀錄；target_id 則代表評論的對象，即買方 / 賣方 / 貼文持有者，用於評估其整體評價紀錄與可信度；rating 則在排序、篩選高低分數評論時提供性能優勢，使平台得以快速呈現優質交易或找出風險帳號。

SQL 建置語法如下：

```
CREATE INDEX IF NOT EXISTS idx_review_order_id ON review(order_id);
CREATE INDEX IF NOT EXISTS idx_review_reviewer_id ON review(reviewer_id
    );
CREATE INDEX IF NOT EXISTS idx_review_target_id ON review(target_id);
CREATE INDEX IF NOT EXISTS idx_review_rating ON review(rating);
```

透過以上索引設計，系統能快速取得訂單的評價記錄、查詢使用者過往評價行為、查看某一帳號的信用評分並依照分數排序評論，不論是買家瀏覽評價或後台稽查帳號，都能擁有更佳查詢體驗與處理效率，並使平台信任機制得以有效運作。

3.3.8 Message 表索引

私訊系統是平台中使用者互動的主要方式之一，message 資料表負責存放雙方溝通內容，包括訊息傳送者、接收者、傳送時間與已讀狀態等欄位。由於訊息查詢行為可能頻繁且即時性需求高，例如顯示聊天室歷史訊息、讀取未讀訊息提醒或依時間排序對話內容，因此建立索引能有效提升整體訊息讀取與傳遞效率。

其中，sender_id 與 receiver_id 用於快速查詢特定使用者之間的訊息往來，當聊天室或對話視窗開啟時，系統可透過索引直接定位相關訊息而避免全表掃描。在通知系統中，is_read 作為未讀訊息篩選條件，索引可使平台快速判斷某位使用者是否有未查看訊息。sent_time 則常用於排序訊息流，如呈現最新對話、顯示時間紀錄、撈取歷史訊息等情境，索引能確保即使訊息量增加仍具備即時讀取能力。

建立索引的 SQL 語法如下：

```
CREATE INDEX IF NOT EXISTS idx_message_sender_id ON message(sender_id);
CREATE INDEX IF NOT EXISTS idx_message_receiver_id ON message(
    receiver_id);
CREATE INDEX IF NOT EXISTS idx_message_is_read ON message(is_read);
CREATE INDEX IF NOT EXISTS idx_message_sent_time ON message(sent_time);
```

透過此索引配置，平台能更迅速完成私訊紀錄讀取、未讀訊息提醒、依時間排序訊息流等任務，並確保聊天系統在大量訊息累積時依然能保持順暢，不會因資料量成長而使訊息開啟延遲或查詢速度下降。

3.3.9 Favorite Posts 表索引

收藏機制讓使用者能快速儲存、追蹤與再次瀏覽感興趣的貼文，而 favorite_posts 資料表即為此功能的核心儲存來源。平台在顯示會員的收藏清單、檢查某篇貼文是否已加入最愛、或分析熱門收藏貼文時，都會頻繁存取此表。因此我們針對常見查詢條件建立索引，以提升系統存取性能。

其中，u_id 用於查詢某位使用者的收藏紀錄。在前端點擊「查看我的收藏」或刷新收藏頁面時，索引能避免逐筆搜索，顯著減少查詢時間。p_id 則可協助後台追蹤某篇貼文的收藏熱度，常用於統計、熱門排序或推薦系統。另一欄位 added_time 則能使系統依收藏時間排序，如顯示最新收藏、歷史追蹤或時間分群分析，索引可避免大量排序運算，進一步降低反應延遲。

建立索引的 SQL 如下：

```
CREATE INDEX IF NOT EXISTS idx_favorite_u_id ON favorite_posts(u_id);
CREATE INDEX IF NOT EXISTS idx_favorite_p_id ON favorite_posts(p_id);
CREATE INDEX IF NOT EXISTS idx_favorite_added_time ON favorite_posts(
    added_time);
```

透過上述索引設計，系統在顯示收藏列表、查詢貼文是否已加入收藏、以及依時間排序收藏紀錄時皆能更快速、低延遲地回應。此外，索引能在資料量累積後維持效能，使收藏功能具備良好擴展性並保持使用體驗流暢。

3.3.10 Posting Images 表索引

貼文圖片為使用者認知商品資訊的重要來源，posting_images 資料表負責儲存貼文對應圖片的檔案位置與顯示順序。由於前端在呈現貼文內容時，會頻繁依照貼文編號載入多張圖片，並依照排序進行顯示，因此建立索引可有效提升貼文圖片讀取速度並降低查詢延遲。

其中，p_id 是最核心的查詢欄位，用於顯示某篇貼文所有圖片。例如使用者點開貼文或切換頁面時，系統會大量依據此欄位查詢，因此建立索引能避免大量 Full Table Scan。另一欄位 display_order 則負責控制顯示排列順序，如縮圖排序、主圖優先顯示等，索引能加速依序排列查找圖片順序的流程，確保商品圖片載入過程更流暢。

SQL 建立索引語法如下：

```
CREATE INDEX IF NOT EXISTS idx_posting_images_p_id ON posting_images(
    p_id);
CREATE INDEX IF NOT EXISTS idx_posting_images_display_order ON
    posting_images(display_order);
```

透過以上索引設定，系統能更快速載入貼文圖片、依照順序呈現圖檔內容，並在貼文圖片量增加後仍維持查詢與排序效率，提升整體瀏覽體驗與資料表的可擴展性。

3.3.11 Course 表索引

課程資料是整個平台分類貼文、搜尋條件與使用者瀏覽邏輯中的關鍵基礎，course 資料表負責儲存科系代碼、分類層級、課程代碼等資訊，並與 posting、favorite_posts、order 等功能相互關聯。因此，我們針對常用的查詢欄位建立索引，使課程搜尋與過濾過程能在大量資料狀況下依然維持快速回應。

其中，dept_id 可用於篩選特定系所課程，在搜尋某系課程或分類貼文時能明顯降低查詢延遲。class_id 則對應課程分類層級，使平台能依班別（Class Category）快速定位課程，常用於分類頁與課程列表展開時。course_code 為課程的唯一識別欄位，使用者在精確查詢或比對課程資料時，索引能避免進行全表掃描，並加速後台比對、資訊串接與貼文標註流程。

SQL 建立索引語法如下：

```
CREATE INDEX IF NOT EXISTS idx_course_dept_id ON course(dept_id);
CREATE INDEX IF NOT EXISTS idx_course_class_id ON course(class_id);
CREATE INDEX IF NOT EXISTS idx_course_code ON course(course_code);
```

透過以上索引機制，課程分類、代碼查詢與系所篩選流程皆能顯著降低讀取成本，並確保課程資料在被頻繁引用至貼文、收藏、訂單與搜尋模組時仍可快速存取，使整體系統的課程導向查詢更加流暢、可擴展且具備長期效能優勢。

3.3.12 索引效果

3.4 交易管理

在本系統中，購買流程被視為一個必須同時滿足原子性（Atomicity）、一致性（Consistency）、隔離性（Isolation）與持久性（Durability）的複合操作。因此，我們選擇將交易邏輯下放到資料庫端，以 PostgreSQL 的 PL/pgSQL 函數實作完整的交易流程，而非單純在應用程式層中依序執行多個獨立的 SQL 指令。在 `005_add_functions.sql` 中，我們定義了一個 `purchase_book(p_buyer_id INT, p_posting_id INT)` 函數，負責處理從檢查商品狀態、驗證餘額、扣款與入帳，到更新刊登狀態與建立訂單的全部步驟。當使用者在前端按下購買按鈕時，系統會呼叫此函數，由資料庫在單一交易（transaction）中完成所有相關更新。

在交易過程中，`purchase_book` 首先透過 `SELECT ... FOR UPDATE` 讀取並鎖定對應的 `posting` 紀錄與買家的 `"user"` 紀錄，確保同一時間只有一個交易流程可以修改同一筆商品與餘額資料，達成併行控制（Isolation）。接著，函

數依序檢查：刊登是否存在、商品狀態是否為可購買的 'listed'、買家是否存在、餘額是否足夠，以及買家與賣家是否為同一人；只要其中任一條件不成立，函數便會立即回傳一個包含 success = false 與錯誤訊息的 JSON，不會對資料庫做任何修改。若所有檢查均通過，函數才會扣除買家餘額、增加賣家餘額、將對應的 posting 狀態更新為 'sold'，並在 orders 表中建立一筆狀態為 'completed' 的訂單，最後回傳包含訂單編號與成交金額的成功結果。

為了讓金流紀錄與商品狀態自動同步，我們在 004_add_triggers.sql 中額外設計了多個與訂單相關的觸發器。例如，update_posting_status_on_order 會在 orders 表新增或更新且狀態為 'completed' 時，自動將對應的 posting 設為已售出；record_transaction_on_order 則會在訂單完成時，同步於 transaction_record 表中新增兩筆紀錄：一筆為買家的付款(負數金額，trans_type = 'payment')，一筆為賣家的收入（正數金額，trans_type = 'income'），形成類似雙向記帳的金流追蹤機制。上述函數與觸發器皆在同一資料庫交易中執行，當 purchase_book 函數內部發生未預期錯誤時，PL/pgSQL 會進入 EXCEPTION 區塊並回傳失敗訊息，該次交易所做的更新將一併回滾（rollback），保證不會出現「扣了錢但沒有訂單」或「商品被標記為已售出但金流未入帳」等不一致情況。透過這樣的設計，我們將購買流程的關鍵不變性約束收斂在資料庫交易中，達成具備 ACID 特性的交易管理。

3.5 並行控制

在本系統中，使用者可能同時對同一筆刊登商品進行購買操作，因此若未妥善處理併行情況，即可能發生「A 與 B 同時購買同一本書，結果兩人都扣款成功」或「帳款已異動但訂單未建立」等資料不一致問題。為避免此類競態條件(Race Condition)，我們將併行控制設計在資料庫層，並以 PostgreSQL 所提供之行級鎖（Row-Level Lock）與交易（Transaction）機制進行保護。

所有購買流程皆由 purchase_book(p_buyer_id, p_posting_id) 函數負責，其程式碼可見於 005_add_functions.sql。當使用者提出購買請求時，系統會先對目標刊登資料執行：

```
SELECT ... FROM posting WHERE p_id = p_posting_id FOR UPDATE;
```

此語句會在資料庫層鎖定該筆刊登紀錄，使得若另一位使用者試圖同時購買同一筆商品，後者操作將被阻塞，直到前一筆交易結束才可繼續。接著系統亦以相同方式鎖定買家餘額之 "user" 資料列，避免同一使用者在多個併行請求下產生「餘額重複扣除」的錯誤。

鎖定完成後，購買函數會在同一交易中依序執行餘額扣款、賣家入帳、更新貼文狀態並建立訂單。若所有動作皆成功，交易即自動提交（Commit）；若任一環節出現錯誤，PL/pgSQL 會進入 EXCEPTION 區塊並中止交易，所有變更將回滾（Rollback），確保不會產生部分更新成功、部分失敗的狀態。透過此併行控制機制，系統可在高併發情境下維持資料一致性與交易正確性。

4 分工資訊

- 資管三 B11705061 羅立宸：
- 資管三 B12705011 黃元翔：
- 資管三 B12705057 陳以倫：

5 專案心得

5.1 資管三 B11705061 羅立宸

5.2 資管三 B12705011 黃元翔

5.3 資管三 B12705057 陳以倫

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	

Table 2: 表 1：資料表 USER 的欄位資訊