資料庫管理（113-1）
作業四

作業設計：孔令傑
國立臺灣大學資訊管理學系

繳交作業時，請至 NTU COOL 下載本作業題目的地方上傳一個 PDF 檔。在生成這個 PDF 檔時，可以用打字的也可以用手寫的，但不管怎樣，請務必注意繳交的文件的專業程度（通常透過排版、文字圖片表格方程式的清晰程度、用字遣詞等面向呈現），如果專業程度不夠，也會被酌量扣分。每位學生都要上傳自己寫的解答。不接受紙本繳交。可以用中文或英文作答。這份作業的截止時間是 ***11 月 20 日早上 08:00:00***，遲交在 12 小時內者扣 10 分、在 12 到 24 小時內者扣 20 分、超過 24 小時的這份作業將得不到分數。

相關規定與提醒

1. 關於上網查詢與 AI 工具：

   任何一份作業都可以被用任何方式完成，包括上網搜尋和使用各種 AI 工具。如果你想用，請留意以下幾件事。首先，抄襲還是不被允許的，如果我們發現抄襲（包括抄襲網路上的答案，或是抄襲同學的答案），都還是會給予嚴厲的懲罰（視情節輕重而定，通常是該份作業算零分，或者不予通過這門課）。只要沒有抄襲，那我們就只根據你交上來的答案的品質給分，不論你是自己想出來的，還是有利用 AI 工具。如果某甲善用了 AI 工具後寫得很好，某乙自己努力寫但寫得不好，那某甲會得到比較高分。其次，如大家所知，AI 工具給的答案可能會錯，也可能不合適。使用 AI 工具是學生的自由，但確認 AI 工具的答案是否合適、是否需要調整則是學生的任務。請務必自行確認答案的正確性與合適性。最後還是想提醒大家，學到多少東西都是自己的，如果一時困難用 AI 工具度過難關那是無妨，但之後建議還是花時間把東西學起來，對自己比較好。

2. 關於「專業」:

   在我們這門課及許多課程中，都需要繳交各種報告。一份報告如果要達成他的效果（例如成功募資、完成溝通的任務等等），除了需要好的內容，也需要「專業」，而顯得專業通常需要「長得好看」以及「看起來用心」，這在沒有標準答案的任務上更是如此。有鑑於此，在這門課的作業和專案，我們都會要求報告的格式和美觀，並且納入評分標準。為此，我們提供報告格式參考指南「DB_reportFormatGuideline.pdf」，上面列舉了一份格式良好的報告的最低標準。在作業一我們會請助教就違反參考指南的地方標出來讓大家知道（我們理解那份指南並不是最完美的，但如果完全沒有標準，同學們容易無所適從，所以我們還是設計一份當標準），但不會扣分，只是提醒大家；從作業二起就會有部分分數是報告專業度分數。之所以要求這些不是想要找大家麻煩，而是大家離出社會也不是太遠了，確實應該要開始被要求報告的可讀性和易讀性，所以我們願意花一些時間要求大家，但不會刁難大家，也請大家理解和盡力嘗試了。

3. 關於「批改」：

   如課程大綱所述，為了不要累死助教，每次作業可能只有部分題目會被批改和給予回饋，但每一題的參考解答都會在作業截止後公佈。如果有一題沒被批改，那所有有寫那一題的學生都會得到那一題的全部分數，但沒寫或期限前沒交作業的自然就不會拿到那一題的分數。最後，請注意是「可能」，換言之也有可能是所有題目都被批改。

題目

1. （20 分）針對上課曾使用過的 `THSR` 資料庫，請考慮以下這句 SQL 指令：

```sql
SELECT Travel_Date, COUNT(*)
FROM Reserved_Ticket
WHERE Depart_Station_ID <= 1030
    AND Arrive_Station_ID = 1035
GROUP BY Travel_Date
HAVING Travel_Date >= '2023-08-02'
```

   注意：本題有許多小題都要求大家去觀察 query plan 和 estimated total cost，但不同電腦、不同 RDBMS 的 estimated total cost 都可能不一樣，甚至 query plan 也可能不一樣。

   (a)（3 分）請簡要地說明這句指令在做什麼。

   (b)（3 分）在沒有任何 index 的情況下，請截圖貼上你的 RDBMS 產出的 query plan。這個 query plan 的 estimated total cost 是多少？

   (c)（4 分）承上題，請觀察這個 query plan 並且找出篩選 `Travel_Date >= '2023-08-02'` 是在哪個階段發生。請用你自己的話說明你的 RDBMS 在那個階段做這個篩選是否合理。

   (d)（4 分）請依序建立三個 single-column index（用預設最陽春的方式建立就好，且不用到 `INCLUDE`），對象分別是 `Depart_Station_ID`、`Arrive_Station_ID` 和 `Travel_Date`。請完整地寫下你建立這三個 index 的語法，然後逐一檢視在只有該 index 存在時的 query plan。請截圖貼上你的 RDBMS 產出的三個 query plan，並且比較它們的 estimated total cost。使用個別 index 得到的 estimated total cost 有沒有大小差別？不論有或沒有，你認為是為什麼？請用自己的話提出解釋。

   (e)（4 分）請幫 `Depart_Station_ID`、`Arrive_Station_ID` 這兩個欄位「同時」建立 multi-column 的 index（用預設最陽春的方式建立就好，且不用到 `INCLUDE`）；由於順序有差，因此你應該可以建立兩個 index。請完整地寫下你建立這兩個 index 的語法，然後逐一檢視在只有該 index 存在時的 query plan（換言之，你應該移除在前一小題建立的 index）。請截圖貼上你的 RDBMS 產出的兩個 query plan，並且比較

它們的 estimated total cost。使用個別 index 得到的 estimated total cost 有沒有大小差別？不論有或沒有，你認為是為什麼？請用自己的話提出解釋。

(f)（2 分）以上我們一共有六個執行該 SQL 指令的方法（沒有 index、三個 single-column index、兩個 multi-column index），請比較這六種方法的 estimated total cost，並且也請在只有各 index 存在時手動執行這句 SQL 指令並且觀察執行時間。estimated total cost 小的 query plan 是否有小的執行時間？不論你觀察到什麼，請用自己的話提出解釋。

2.（20 分）請考慮以下數列

$$200, 800, 300, 400, 900, 600, 100, 350, 375, \boxed{360}, 500, 850, 950, 970, 920, 940, \boxed{930}。$$

(a)（3 分）把這個數列由左到右依序插入到一個原本是空的、一個 inner node 最多向下分支數為 4、一個 leaf node 最多存 3 個 key 的 B+ tree 中，並且把最終結果畫出來。你可以用老師上課教的那個網頁沒問題，也可以在完成後直接螢幕截圖做繳交。

(b)（3 分）請用你的話解釋，插入數列中框起來的那個 360 的時候，這棵 B+ tree 是怎麼調整它自己的。你可以把插入 360 前和後甚至過程中的 B+ tree 畫出來，方便你做解釋。你可以螢幕截圖，但你必須用自己的話說明這個流程。

(c)（4 分）請用你的話解釋，插入數列中框起來的那個 930 的時候，這棵 B+ tree 是怎麼調整它自己的。你可以把插入 930 前和後甚至過程中的 B+ tree 畫出來，方便你做解釋。你可以螢幕截圖，但你必須用自己的話說明這個流程。

(d)（5 分）請把這個數列排序，然後用 bottom-up 的方式建構一棵 B+ tree，這棵 B+ tree 的 inner node 的 order 為 4（最多指向 4 個 child node）、leaf node 的 order 為 3（每個 leaf node 最多放 3 個 key 值）。建構時，請讓每個 leaf node 一律都含有 2 個元素（但如果數列中的數字有奇數個，則最後一個 leaf node 裡面請放 3 個元素），而除了最後面的 inner node 可以只指向 2 個 child node，前面的每個 inner node 都是指向 3 個 child node。請把完成後的完整的 B+ tree 畫出來。

(e)（5 分）請問用 bottom-up 的方式建構的 B+ tree 跟用 top-down 方式從小到大依序插入的 B+ tree，會長得一樣還是不一樣？為什麼？請用自己的話說明。

注意：這樣的題目在期末考也可能會出現。我們固然接受大家在寫作業時用工具輔助作答，但請大家確保自己確實理解觀念（進而在只有紙筆的環境下也有辦法作答）。

3.（10 分）針對上課使用的 THSR 資料庫，請寫一個合情合理的 SQL 查詢，去展示使用 index 與否的差異（當然，我們是期待你展示有用有差）。你應該敘述你的 SQL 查詢指令、你建立 index 的指令、使用 index 前後的 query plan 及其差異，以及使用 index 前後的運算時間差異。你的 SQL 指令應該要有 join 至少兩張表。

4.（15 分）在課程中教了三種 nested loop join algorithm，包含 stupid nested loop join、single-block nested loop join，以及 multi-block nested loop join。假設現在有兩個資料表

R 跟 S，分別有 $m = 500000$ 和 $n = 4000000$ 筆資料，分別放在 $M = 5000$ 和 $N = 40000$ 個 page 中。假設某台電腦上讀或寫一個 page 所需的時間是 0.1 ms，有 $B = 100$ 個 buffer page 可以用。

(a)（10 分）請分別計算對 R 和 S 執行這三種 nested loop join algorithm 所需的時間。

(b)（5 分）請用自己的話說明為什麼前一小題的三種演算法會有效率上的差異。

5.（20 分）在我們上課教過的 `COMPANY` 資料庫中有三張表 `EMPLOYEE`、`DEPARTMENT` 和 `DEPT_LOCATIONS`。我們可以用以下 SQL 指令把這三張表 join 起來：

```
SELECT e.Fname, e.Lname, e.Dno, d.Dname, dl.Dlocation
FROM EMPLOYEE AS e
    JOIN DEPARTMENT AS d ON e.Dno = d.Dnumber
    JOIN DEPT_LOCATIONS AS dl ON d.Dnumber = dl.Dnumber
```

(a)（3 分）請用自己的話說明這句 SQL 指令在做什麼、會回傳怎樣的資料。請執行這句 SQL 指令，並且截圖貼上回傳的資料中的前五筆。

注意：每個人看到的前五筆的內容可能並不相同，但格式、意涵應該是相同的。

(b)（4 分）請使用上課教的樹狀圖表示法（類似圖 1 這種），列出所有把這三張表 join 起來的 query plan。在回答這一題時，請不用在意一個 join 的兩張表誰當 outer table 誰當 inner table，亦即 A join B 跟 B join A 可以被視為是相同的，所以圖 1 中的左邊兩棵樹我們視為相同、右邊兩棵樹我們視為相同，甚至這四棵樹都是相同的，所以你的答案應該只有 3 棵樹。

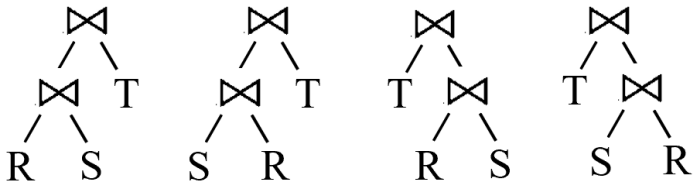

圖 1: 範例樹狀圖表示法

(c)（4 分）承上題，請先讓你的 query planner 更新他的統計資料（在 PostgreSQL 就是執行 `ANALYZE`），然後讓 query planner 產出 query plan。請截圖貼上你的 RDBMS 為這個 SQL 指令產出的 query plan，並且說明這是你前一題的樹狀圖的哪一個。接著請用你自己的話說明 query planner 在六棵樹中選擇這棵樹可能是基於什麼原因，以及你覺得是否合理。在回答時，請考慮各資料表的資料筆數與各種 join 完之後會得到的資料筆數，看看這些資訊對你的回答是否有幫助。

(d)（4 分）承上題，假設要被執行的 SQL Select statement 是

```sql
SELECT ed.Fname, ed.Lname, ed.Dno, ed.Dname, dl.Dlocation
FROM
(
    SELECT e.Fname, e.Lname, e.Dno, d.Dname
    FROM EMPLOYEE AS e
        JOIN DEPARTMENT AS d ON e.Dno = d.Dnumber
) AS ed
    JOIN DEPT_LOCATIONS AS dl ON ed.Dno = dl.Dnumber
```

請看看你的 query planner 產出的 query plan 跟前一小題的相同還是不同。如果不同，請截圖貼上你看到的 query plan 並且用自己的話說明不同的原因；如果相同，就不用再次截圖貼上 query plan，但請還是用自己的話說明為什麼會相同。

6.（15 分；每小題 5 分）針對上課教過的 `THSR` 資料庫，你想要執行以下這句 SQL 指令

```sql
SELECT rt.arrive_station_id, m.name
FROM reserved_ticket AS rt
    JOIN member AS m ON rt.citizen_id = m.citizen_id
WHERE rt.travel_date = '2023-08-01'
    AND rt.depart_station_id <= 1000
    AND m.name LIKE '% A%'
```

在本題中，讓我們試著自己實做 Hash join，流程如下：

- **步驟 1**：請執行

```sql
SELECT arrive_station_id, citizen_id
FROM reserved_ticket
WHERE travel_date = '2023-08-01'
    AND depart_station_id <= 1000
```

以得到初步的 21290 筆資料。

- **步驟 2**：請寫一個 hash function，把步驟 1 得到資料放進 100 個 bucket 裡，而 hash function 的運作方式是把 `citizen_id` 的每個字元的 ASCII code 加總後除以 100 得到的餘數，其中數字字元也要被轉成 ASCII code。舉例來說，「AAAAW1776」的 ASCII code 加總是 $65 + 65 + 65 + 65 + 87 + 49 + 55 + 55 + 54 = 560$，因此「AAAAW1776」應該被放進代表餘數為 60 的 bucket 裡。在每個 bucket 存放一筆資料時，請將該筆資料的 `arrive_station_id` 和 `citizen_id` 都存起來，並且新增一個空的欄位等等要用。
- **步驟 3**：請執行

```
SELECT citizen_id, name
FROM member
WHERE name LIKE '% A%'
```

並且得到 10248 筆資料。

- **步驟 4**：請用前述的 hash function，把步驟 2 得到的資料的 `citizen_id` 逐一放進 100 個 bucket 裡，並且在該 bucket 裡面搜尋是否有 `RESERVED_TICKET.citizen_id` 跟我們手上這筆資料的 `MEMBER.citizen_id` 資料相符，如果有就把我們手上這筆資料的 `MEMBER.name` 放進那些（個）`RESERVED_TICKET` 資料的第三個欄位。
- **步驟 5**：從第 1 個到第 100 個 bucket，依序檢視每個 bucket 裡的每筆資料，如果該筆資料的 `name` 是 NULL 那就跳過，不然就將該筆資料的 `arrive_station_id` 和 `name` 印出。

請回答以下問題：

(a) 請用你喜歡的程式語言實作前述 hash join 演算法，然後複製貼上你的完整程式碼，並且適度地加上註解。請實際執行你的演算法，並寫下代表餘數為 60、70 和 80 的 bucket 裡，各有幾筆 `RESERVED_TICKET` 資料。

(b) 我們也可以用很暴力的方式實現這個 join，就是把步驟 1 得到的 21290 筆資料和步驟 3 得到的 10248 筆資料直接交叉比對，意即寫一個雙層迴圈，對於 10248 筆資料的每一筆，我們都掃一遍 21290 筆資料，去幫一些資料填入 `name` 欄位的值，等雙層迴圈結束後，再掃一遍 21290 筆資料，把 `name` 不是 NULL 的資料的 `arrive_station_id` 和 `name` 印出。請用你喜歡的程式語言實作這個暴力 join 演算法，然後複製貼上你的完整程式碼，並且適度地加上註解。

(c) 請實際執行你實作的兩個演算法，並比較前面的 hash join 演算法和這個暴力 join 演算法「在實際執行 join 部份」的執行時間（意即不要把前面讀取資料、後面印出結果的時間計入）。請寫下兩個演算法各需要多長的執行時間來完成任務，並且用自己的話說明為什麼某個演算法比另一個快（或者都差不多）。不論你觀察到的執行時間為何，hash join 相較於暴力 join 有什麼缺點？請用你自己的話說明。