資料庫管理（114-1）
作業五

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；不接受遲交。可以用中文或英文作答。這份作業的截止時間是 ***12 月 3 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. （10 分）在課程中教了三種 nested loop join algorithm，其中一種是 multi-block nested loop join。我們說如果有 B 個 buffer page 可以用的話，應該一次讀入 $B-2$ 個 outer table 的 block 和 1 個 inner table 的 block。請問如果改成「一次讀入 1 個 outer table 的 block 和 $B-2$ 個 inner table 的 block」，效率會如何？和本來會大致相同還是不同？你可以假設 outer table 和 inner table 各有 m 和 n 筆資料、各存放在 M 和 N 個 disk block 上，去做相對應的計算、討論和說明。

2. （20 分；每小題 5 分）在我們上課教過的 `RETAIL` 資料庫中，我們可以執行以下查詢：

```
SELECT sd.product_id, p.name, s.date_time,
       sd.unit_price, sd.qty
FROM SALES_DETAIL AS sd
    JOIN SALES AS s ON sd.receipt_no = s.receipt_no
    JOIN PRODUCT AS p ON sd.product_id = p.id
WHERE s.store_id = 1
    AND sd.unit_price >= 20
```

   讓我們假設 `PRODUCT`、`SALES` 和 `SALES_DETAIL` 各有 100 筆、1000 筆和 5000 筆資料。

(a) 我們正要生成一個查詢計畫（query plan），需要合併這三張表。有人跟你說，原則上應該要盡量先合併小的表，並且在合併兩張表時把較小的表放在左側，因此他打算先合併 PRODUCT 和 SALES（PRODUCT 在左），再把合併後的結果跟 SALES_DETAIL 合併（SALES_DETAIL 在右）。你同意嗎？你認為這個查詢計畫是有效率的嗎，為什麼？

(b) 我們正要生成一個查詢計畫（query plan），會先合併 SALES 和 SALES_DETAIL（SALES 在左），再把合併後的結果跟 PRODUCT 合併（PRODUCT 在左）。在此架構下，請使用上課教的樹狀圖表示法（類似圖 1 這種），去展示你會如何把 predicate pushdown 和 projection pushdown 做到極致。

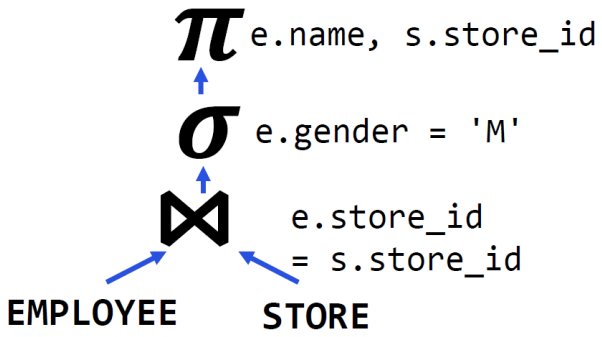

圖 1: 範例樹狀圖表示法

(c) 請忽略前兩小題，直接到你的 DBMS 去檢視該 DBMS 生成的查詢計畫（query plan），將該計畫（包含其估計的成本）截圖貼上，並用自己的話說明為什麼該 DBMS 會覺得那個計畫是有效率的。

(d) 假設要被執行的查詢被換成

```
SELECT sdp.product_id, sdp.name, s.date_time,
       sdp.unit_price, sdp.qty
FROM SALES AS s
    JOIN
    (
        SELECT *
        FROM SALES_DETAIL AS sd
            JOIN PRODUCT AS p ON sd.product_id = p.id
    ) AS sdp ON s.receipt_no = sdp.receipt_no -- AND sd
       .product_id = sp.id
WHERE s.store_id = 1
    AND sdp.unit_price >= 20
```

請看看你的 query planner 產出的查詢計畫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。不論你的答案是什麼，請特別說明 DBMS 有或沒有先做子查詢再完成整句查詢、在什麼時機做篩選，以及你覺得 DBMS 那麼做是否合理。

3.（25 分）針對上課教過的 THSR 資料庫，你想要執行以下這句 SQL 指令

```
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

  ```
  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 $N = 500$ 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 N 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 N 個 bucket 裡，並且在該 bucket 裡面搜尋是否有任何一筆資料的 `RESERVED_TICKET.citizen_id` 跟我們手上這筆資料的 `MEMBER.citizen_id` 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` 資料的那個空欄位。
- **步驟 5**：從第 1 個到第 N 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)（10 分）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 0、1 直到 99 的 bucket 裡，一共有幾筆 `RESERVED_TICKET` 資料。

(b)（10 分）我們也可以實作 sort-merge join，就是把步驟 1 得到的 21290 筆資料和步驟 3 得到的 10248 筆資料個別排序，然後按順序檢視並合併兩批資料，最後把 `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)（5 分）請實際執行你實作的兩個演算法，並比較你的 hash join 演算法和 sort-merge 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 sort-merge join 有什麼缺點？請用你自己的話說明。

4.（10 分）請用你自己的話說明 ACID 和 BASE 的差別，並且各舉一個例子說明哪種應用應該遵循 ACID，哪種則應該遵循 BASE。

5.（15 分；每小題 5 分）NoSQL 和資料庫分片（sharding）是在處理超大量資料時人們會考慮的技術。關於 NoSQL 和分片，請用你自己的話回答以下問題：

(a) 某公司對許多企業提供商情調查的資訊服務，服務內容如下。首先，該公司製作一個免費手機應用程式，讓一大堆消費者下載使用以管理自己的發票（登錄、自動對獎、消費紀錄分析等等）。等有夠多消費者使用、累積夠多張發票之後，該公司就開放企業客戶來購買商情（由發票彙總的商業資訊），

例如讓麥當勞知道特定地區的肯德基生意如何、讓星巴克知道某特定商圈的消費力等等。

此公司苦於資料量太多（收著收著就有數千萬張發票了），因此使用了 NoSQL 而非關聯式資料庫來儲存這些資料。你認為合理嗎？為什麼？請從此應用的性質與關聯式資料庫和 NoSQL 的性質進行分析。

(b) 承上題，此公司希望用分片方式來進一步提升效率。請建議他們該如何分片（例如按客戶分、按時間分、按地區分、用 hash 分等等），並說明原因。

(c) 請考慮你的期末專案的主題。假設在該應用中你要做分片，你會對哪些（個）資料表做分片，你又會怎麼分[1]？為什麼？

6.（20 分；每小題 5 分）上課介紹過的 RETAIL 資料庫中有以下兩個資料表：

```
PRODUCT(id, name)
SALES(receipt_no, date_time, store_id)
SALES_DETAIL(receipt_no, product_id, unit_price, qty)
```

讓我們假設 PRODUCT 的兩個欄位依序各佔 4 位元組和 50 位元組，SALES 的三個欄位依序各佔 10 位元組、8 位元組與 4 位元組，而 SALES_DETAIL 的四個欄位依序各佔 10 位元組、4 位元組、4 位元組與 4 位元組。PRODUCT、SALES 和 SALES_DETAIL 各有 100 筆、1000 筆和 5000 筆資料，其中 SALES 的資料中有 400 筆發生在編號為 1 的門市、SALES_DETAIL 的資料中有 1000 筆發生在編號為 1 的門市。

有一家公司把這三張表存在一個分散式資料庫中，其中 PRODUCT、SALES 和 SALES_DETAIL 各存在地點 1、地點 2 和地點 3。這家公司現在在地點 2 執行以下查詢：

```
SELECT sd.receipt_no, sd.product_id, p.name,
       sd.unit_price * sd.qty AS sales_amt, s.date_time
FROM SALES_DETAIL AS sd
    JOIN SALES AS s ON sd.receipt_no = s.receipt_no
    JOIN PRODUCT AS p ON sd.product_id = p.id
WHERE s.store_id = 1
```

針對以下每個查詢計畫（query plan），請計算該查詢計畫的總資料傳輸量（以位元組為單位），並附上簡要的計算過程與說明。如果有複數種滿足該查詢計畫的

[1]雖然你在期末專案會用關聯式資料庫，但你還是可以考慮分片（sharding）；你可以想成是你改用了 NoSQL 資料庫，或者你使用了關聯式資料庫的分片技術。總之，請不要在意「關聯式資料庫相對較不易在分散式架構運行」這個事實。

作法，請使用讓總傳輸量最小的作法；如果你使用的不是讓總傳輸量最小的作法，會被酌予扣分。

(a) 把 PRODUCT 中的必要資料傳到地點 2、把 SALES_DETAIL 中的必要資料傳到地點 2，然後在地點 2 做合併和篩選。

(b) 把 PRODUCT 中的必要資料傳到地點 3、把 SALES 的必要資料傳到地點 3、在地點 3 做合併，然後把合併結果傳到地點 2，最後在地點 2 做篩選。

提示：在把篩選過的 SALES 的必要資料傳到地點 3 時，可以不用把 date_time 傳過去。

(c) 把 PRODUCT 的必要資料傳到地點 3、把篩選過的 SALES 的必要資料傳到地點 3、在地點 3 做合併，然後把合併結果傳到地點 2。

(d) 把 PRODUCT 的必要資料傳到地點 2、把篩選過的 SALES 的必要資料傳到地點 3、在地點 3 合併 SALES 和 SALES_DETAIL，然後把合併結果傳到地點 2，最後在地點 2 再做一次合併。