6.5830/31 Database Systems: Fall 2023 Quiz I

There are 13 questions and 14 pages in this quiz booklet. To receive credit for a question, answer it according to the instructions given. You can receive partial credit on questions. You have 80 minutes to answer the questions.

Write your name on this cover sheet AND at the bottom of each page of this booklet.

Some questions may be harder than others. Attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can’t understand your answer, we can’t give you credit!

THIS IS AN OPEN BOOK, OPEN NOTES QUIZ. LAPTOPS MAY BE USED FOR NOTES AND SLIDES; NO PHONES, INTERNET, NOR ON LAPTOP LLMS, SQL SHELLS, NOR OTHER ASSISTANTS.

Name:
I  Cost Model

Assume a row-oriented disk-based database with the following tables:

CREATE TABLE airport{
    code CHAR(4) PRIMARY KEY,
    name CHAR(115) NOT NULL,
    country-code CHAR(2) NOT NULL
}

CREATE TABLE flight{
    nb CHAR(8) PRIMARY KEY,
    origin CHAR(4) REFERENCES airport(code) NOT NULL,
    destination CHAR(4) REFERENCES airport(code) NOT NULL
}

CREATE TABLE passenger{
    pid INT PRIMARY KEY,
    name CHAR(105) NOT NULL,
    flight-nb CHAR(8) REFERENCES flight(nb) NOT NULL
}

The airport table has 1K records, flights has 1M records, and passengers table 100M records and every flight has exactly 100 passengers.

All the data is organized as a heap file with a page size of 64KB, all fully filled except potentially the last one. While each record has a header (e.g., for the rid) the heapfile itself does NOT have any additional meta-data/header. Chars are 1 byte and ints are 8 bytes.

The heap file is stored on a spinning disk with the following characteristics:

Seek: 10ms

Bandwidth: 100MB/sec

Throughout this exam we use KB = 1000 bytes, MB = 1000 KB, etc.
1. **[9 points]**: Calculate (1) the number bytes for each record in the respective tables assuming a 4 byte header per record, (2) the time to scan the tables including seek, and (3) the number of pages (assume all pages are dense except potentially the last one). Write your answer into the following table.

<table>
<thead>
<tr>
<th>Table</th>
<th>Bytes</th>
<th>Scan-time</th>
<th># pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>airport</td>
<td>Answer: 125</td>
<td>Answer: 0.01125s and 0.01128s both accepted</td>
<td>Answer: 2</td>
</tr>
<tr>
<td>flight</td>
<td>Answer: 20</td>
<td>Answer: 0.21s and 0.21032s both accepted</td>
<td>Answer: 313</td>
</tr>
<tr>
<td>passenger</td>
<td>Answer: 125</td>
<td>Answer: 125.01s and 125.01032s both accepted</td>
<td>Answer: 195313</td>
</tr>
</tbody>
</table>

Name:
2. [5 points]:
Consider the following query:

```sql
SELECT p.name, a.code
FROM airport a, flight f, passenger p
WHERE p.flight-nb = f.nb
AND f.origin=a.code
AND f.flight-nb in ('TP1023', 'TP343', 'TP225', 'TP3423', 'TP107', 'LH130', 'LH1034', 'UA455', 'UA230', 'UA260')
LIMIT 10
```

(Fill in the blanks to indicate the output cardinality of each intermediate result in the following physical execution plan for this query)

```
Answer: Filter = 10, Join1 = 1000, Join2 = 1000, Projection = 1000, Limit = 10
```
3. [10 points]:
What is the estimated execution time considering only I/O for the query plan above assuming a cold (empty) cache, a 2000 page cache (with LRU), and that each operator computes and stores the entire (intermediate) result in memory before the next operation begins? That is, the operators do NOT implement an iterator tuple-at-a-time model. [You may round results up/down to second granularity]

(Write your answer in the space below.)

Answer: 1250s

4. [6 points]: Assume the query plan above with the same assumptions (cold cache etc.), but this time we execute it using a tuple-at-the-time iterator model (like in GoDB). Would the query execute faster or slower?

(Write your answer and short explanation below.)

Answer: Faster since we only need to compute 10 tuples.
5. **[8 points]**: From the list below, select all *unclustered* indexes which would help to improve the execution time of the query above (the indexes may result in a different physical plan). Provide a brief explanation of why you selected the indexes you did:

A. Index on airport(code)
B. Index on airport(name)
C. Index on flight(nb)
D. Index on flight(origin)
E. Index on passenger(pid)
F. Index on passenger(flight-nb)

**Explanation:**

*Answer: C and F would help: C speeds up the filter so we don’t need to scan the entire flights table. F lets us do an index nested loop join for join1.*

6. **[4 points]**: Assume the query plan above with the same assumptions (cold cache etc.), but this time we execute it using an early materialization column-store design without compression. Would the query execute faster or slower?

*(Write your answer and short explanation below.)*

*Answer: Faster since not all columns need to be read.*
II Iterator Model

7. [10 points]:
Consider a database that implements a tuple-at-a-time iterator model as in GoDB. Which of the following statements are true?

(Circle ‘T’ or ‘F’ for each choice.)

T F The iterator model can help to minimize the amount of memory needed for temporary results
T F The iterator model can cause a lot of (virtual) function calls per tuple, which are hard to optimize for modern compilers
T F The iterator model might fetch fewer pages from disk than alternative execution models (e.g., computing one operator at a time)
T F The iterator model helps to take full advantage of the CPU cache for queries that operate on large amounts of data with many joins
T F The iterator model allows for a lot of parallelism and can be easily SIMD-optimized

Answer: TTTFF
8. [10 points]: Consider the following iterator with the limit operator at the root and scan operators at the bottom. Here we use the same schema for the airports database as above. The numbers to the right of each operator indicate cardinality estimates (e.g., the airport scan yields 1k tuples). The join between flights and airports is on the *origin* column and every flight has exactly one airport it originates from. Assume the iterator tree implements the tuple-at-a-time next() API and that the scan operator internally uses a shared buffer manager that also implements the next() API to get tuples from a particular page (like the GoDB implementation). Here, “build-side” means the relation the hash table is built on.

How many next() invocations are required until the query is completely computed? Note, that each next() call on every operator counts as one invocation. For example, if projection.next() calls filter.next() this counts as two invocations.

(Write your answer below.)

Answer: 2050.

Let’s first consider what happens on the very first invocation:

A. The first invocation of next() is on limit (1)
B. Limit then calls next() on project (1)
C. Project calls next() on the in-memory hash join (1)
D. The in-memory hash join has to fully create the build-side and then calls next() on the probe side. That is:

   (a) Build-side: it calls next() on the scan, which calls next() again on the buffer-pool/page (2 next() calls per invocation). Given that we have 1k records, we call next() 2,000 times to create the build-side (2,000).

   (b) Probe-side: After creating the hash table, we have to read a record from the probe side. These are again 2 next() calls (2).
So to get the first record we do a total of $1 + 1 + 2000 + 2 = 2005$ next() calls.

However, the second record takes much less:

A. *Invocation of next() on limit (1)*

B. *Limit then calls next() on project (1)*

C. *Project calls next() on the in-memory hash join (1)*

D. *The in-memory hash:

   (a) Build-side: is already done (0)

   (b) Probe-side: again 2 next() calls to read a record from the page (2)*

To get the second or any other record we only require $1 + 1 + 0 + 2 = 5$ next() calls. However, after the limit operator returns 10 records, we are done. This gives us a total of $2005 + 9 \times 5 = 2050$ invocation.

Note: If we want to be really precise we would have some additional next calls that return “nil”. For example, after calling next() 10 times on limit we would call it one more time and receive “nil”, which indicates that there are no more records. Similarly the scan of the build side will do this for each page, and then the scan will also return “nil” at the end. When grading the solution we accepted solutions which considered the extra next() calls for “nil” and solutions which did not.
### III Optimization

Consider the following database about animals, the hats they wear, and the magical effects of these hats.

- **animals** (AId, AName, ASpecies)
- **hats** (HId, HColor, HSize)
- **wears** (W_AId, W_HId)
- **magic** (MId, MDesc)
- **hatmagic** (HM_HId, HM_MId)

Here **underlined** fields are a part of the primary key and **italic** fields are a foreign key reference.

You want to find the magic that the animal "Tim the Beaver" gets, e.g. evaluate this query:

```sql
SELECT MDesc FROM animals A JOIN wears W ON AId = W_AId
JOIN hats H ON HId = W_HId
JOIN hatmagic HM ON HM_HId = HId
JOIN magic M ON MId = HM_MId
WHERE AName = "Tim the Beaver"
```

9. **[6 points]**: Suppose the Selinger optimizer chooses to compute the join between animals and wears first. Complete the following query plan with one plan the Selinger optimizer might generate (you can choose any plan it would produce). You are given no information about table sizes or join selectivities. Fill in the blanks with a join symbol (▷ ◁) or the abbreviation of a table (HM, H, M). You will need to leave some blanks empty.

```
Answer: A left deep plan that first joins with H, then HM, and finally M.
```

Name:
IV ER Diagrams

Consider the following ER diagram

Here underlined names are primary keys. Because the N:1 notation can be confusing, consider this example, which means “there are many fish, but each is in one bowl”.

10. [6 points]: Which statements about a database that follows the constraints shown in the ER diagram can be true:

(Circle ‘T’ or ‘F’ for each choice.)

T F In the same semester, one student can take multiple classes, and one class can be taken by multiple students

T F One class can be in multiple departments

T F One professor can be in multiple departments

T F One professor can teach multiple classes in a semester

T F In a semester, a class’s rating is independent of the professor who taught it

T F A professor’s rank depends on the department they are in

Name:
Answer: TFFTFF

11. [10 points]: Which of the following table definitions would exist in the schema for this diagram. If you circle false, write a brief explanation for why not.

T  F  Class : (CName, CNumber, CDescription, CProfessor)
Explanation if false:

T  F  Student : (KerblId, Name)
Explanation if false:

T  F  Professor: (PName, PRank, PDepartment)
Explanation if false:

T  F  ClassIn: (CName, DName, Semester)
Explanation if false:

T  F  Takes(KerblID, CName)
Explanation if false:

Answer: FFTTFF.
The explanation for the first question: the class can be taught by multiple professors, so one “CProfessor” column can not represent this information.
The explanation for the fourth question: “semester” is not an attribute or “ClassIn” relation.
The explanation for the fifth question: the table “Takes” misses “Grade” and “Semester” columns, which are attributes associated with “Takes” in the ER diagram.
V  SQL

Consider the following database representing people and the dorms they live in:

Dorm (DormID, Name, Address)
DormRooms (RoomId, RoomNo, DRDormId, SqFt, Beds)
Student (KerbId, SName, SRoomId)
ConnectsTo (RoomIdA, RoomIdB)

Here underlines represent primary key attributes, and italics represent foreign key attributes. ConnectsTo denotes that two rooms are connected by a door or corridor. Some rooms may have 0 beds, indicating that they are common areas. For convenience the ConnectsTo relation duplicates each connection in the opposite direction (e.g., if there is an entry “1,2”, there is also an entry “2,1”).

Complete the following queries:

12. [8 points]: Find students who do not have a dorm room:

SELECT KerbId FROM
Student ON SRoomId = RoomID
WHERE

Answer:

SELECT KerbId FROM
Student LEFT JOIN / OUTER JOIN DormRooms ON SRoomId = RoomID
WHERE RoomID IS NULL

13. [8 points]: Find the total square feet of all rooms connected to room 27:

SELECT FROM
DormRooms dr

Name:
WHERE dr.RoomID = 27

Answer:

SELECT SUM(dr2.SqFt) FROM
DormRooms dr JOIN ConnectsTo ON dr.RoomId = RoomIdA
JOIN DormRooms dr2 ON dr2.RoomId = RoomIdB
WHERE dr.RoomID = 27

Alternative Answer:

SELECT SUM(dr2.SqFt) FROM
DormRooms dr, ConnectsTo, DormRooms dr2
WHERE dr.RoomID = 27 AND dr.RoomId = RoomIdA AND dr2.RoomId = RoomIdB

End of Quiz I!