UCLA
Computer Science Department
Fall 2021

Instructor: J. Cho

CS143 Quiz 2: 2 hours

Student Name: ____________________

Student ID: ____________________

(IMPORTANT PLEASE READ **):**

- The exam is *closed book* and *closed notes.* You may use *two double-sided cheat-sheets*, 4 pages in total. You can use a calculator.
- *Simplicity and clarity of your solutions will count.* You may get as few as 0 point for a problem if your solution is far more complicated than necessary, or if we cannot understand your solution.
- If you need to make any assumption to solve a question, *please write down your assumptions.* To get partial credit, you may want to write down how you arrived at your answer step by step.
- If a question asks for a numeric answer, you don't have to calculate. You may just write down a numeric expression.
- Please, write your answers neatly.

Problem	**Score**	
1	20	
2	20	
3	20	
4	20	
5	10	
6	10	
Total	100	

Problem 1: Database Integrity (20 points)

For all questions in this problem, we refer to three tables created by the following statements:

```
CREATE TABLE Studio(
    id      INT,
    name    VARCHAR(50),
    PRIMARY KEY(id),
    UNIQUE(name)
);

CREATE TABLE Movie(
    id      INT,
    title   VARCHAR(50),
    year    INT,
    sid     INT,
    PRIMARY KEY(id),
    FOREIGN KEY(sid) REFERENCES Studio(id)
        ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE ShowTime(
    theater     VARCHAR(50),
    mid         INT,
    time        DATETIME,
    ticketPrice DECIMAL(5,2),
    PRIMARY KEY(theater, mid),
    FOREIGN KEY(mid) REFERENCES Movie(id)
        ON UPDATE CASCADE
);
```

As you can imagine from their names, `Studio` table contains the movie production studio information (like "Disney"), `Movie` table contains the movie information (like "Avengers: Endgame"), and the `ShowTime` table contains the movie showtime information at theaters.

1. Assume that the production studio of a movie never changes once this information is inserted into the three tables. Given this, is it possible to use a SQL92 CHECK constraint to enforce that the ticket price of any Disney movie should never be below \$5.00? If your answer is yes, write down the CHECK constraint and the name of the table to which the constraint should be added. If your answer is no, briefly explain why it is not possible. (8 points)

   ANSWER:

   YES. The following constraint can be added to the `ShowTime` table

   ```
   CHECK( 'Disney' <> (SELECT S.name
                       FROM Studio S, Movie M
                       WHERE S.id = M.sid AND M.id = ShowTime.mid)
          OR ticketPrice >= 5.00 )
   ```

2. For this problem, consider the three tables created with the earlier `CREATE TABLE` statement, but ignore any assumption that we made in P1.1 The three tables currently have the following tuples:

   Studio

id	name
1	Disney
2	Universal
3	20th Century

   Movie

id	title	year	sid
1	Black Panther	2018	1
2	Minions	2015	2
3	Frozen	2013	1
4	Avatar	2009	3

   ShowTime

theater	mid	time	ticketPrice
AMC	1	2021-11-25 10:00:00	10.00
Regency	3	2021-11-24 21:00:00	9.00
Landmark	3	2021-11-23 20:00:00	11.00
AMC	4	2021-11-27 09:00:00	7.00

   Now, for the statement(s) given on the following page, briefly state what will happen when the statements are issued in the given sequence. If some tuples are inserted, updated, or deleted due to the statements, clearly indicate the exact list of tuples that are affected (and their new values if they are updated). If any statement is rejected, briefly explain why. Consider each subproblem separately from others. Each of the following subproblems amounts to 3 points.

1.
   ```
   INSERT INTO Movie  Values (5, 'Spiderman', 2012, 4);
   INSERT INTO Studio Values (4, 'Sony');
   ```

   ANSWER:
   The first statement is rejected. The inserted Movie tuple references sid 4, which does not exist in the Studio table. If the order of the INSERT statements were reversed, both tuples would have been inserted.

2. `DELETE FROM Studio WHERE id = 2;`

   ANSWER:
   (2, 'Universal') is deleted from the Studio table. (2, 'Minions', 2015, 2) is also deleted from the Movie table due the "ON DELETE CASCADE" clause of the FOREIGN KEY constraint of the Movie table.

3. `DELETE FROM Studio WHERE name = '20th Century';`

   ANSWER:
   The statement is rejected. The Movie tuple (4, 'Avatar', 2009, 3) references the deleted studio tuple with the "ON DELETE CASCADE" clause. So DBMS will try to delete the movie tuple as well. However, this movie tuple is referenced by the ShowTime tuple ('Fox', 4, '2021-11-27 09:00:00', 7.00) and will become dangling if the movie tuple is deleted. Since the ShowTime FOREIGN KEY constraint does not have an ON DELETE clause, the system will not handle this violation and reject the statement.

4. `UPDATE Movie SET id = 5, sid = 1 WHERE id = 4;`

   ANSWER:
   The Movie tuple (4, 'Avatar', 2009, 3) will be updated to (5, 'Avatar', 2009, 1) and the ShowTime tuple (AMC, 4, '2021-11-27 09:00:00', 7.00) is updated to (AMC, 5, '2021-11-27 09:00:00', 7.00).

Problem 2: Disks and Files (20 points)

Consider a movie table created by the following SQL statement:

```
CREATE TABLE Movie(
    id        INT,
    title     CHAR(50),
    studio    CHAR(50),
    year      INT
)
```

An integer value takes up 4 bytes to store and `CHAR(n)` type takes n bytes. The Movie table contains 10,000 tuples and is stored as fixed-length records. The tuples are stored in a row-oriented format and are *not spanned*. We use a disk of the following parameter to store the table.

- 4 platters (8 surfaces)
- 10,000 tracks
- 1,000 sectors per track
- 4,096 bytes per sector
- 6,000 RPM rotational speed
- 10ms average seek time, 2ms minimum seek time between adjacent tracks, 30ms maximum full-stroke seek time

Assume that all blocks for the table are all allocated sequentially. That is, if the table fits in one track, all blocks for the table are allocated as consecutive blocks on the same track. If the table size is bigger than one track, the blocks are allocated in multiple adjacent tracks. Assume that one disk sector corresponds to one disk block.

1. What is the minimum number of blocks that we need to store the movie table? (5 points)

   ANSWER:
   The size of each tuple is $4 + 50 + 50 + 4 = 108$ bytes. The size of each block is 4096. Because tuples are not spanned, we can store $\lfloor \frac{4096}{108} \rfloor = 37$ tuples per block. Since we have 10,000 tuples, we need $\lceil \frac{10{,}000}{37} \rceil = 271$ blocks to store the entire table.

2. For this subproblem, consider the following SQL query:

```
SELECT * FROM Movie WHERE year = 2020 FETCH FIRST 1 ROWS ONLY;
```

Assume that the table has not been sorted by any attribute and no index has been constructed on the table. Therefore, the system executes the above query by scanning the table from the beginning until a matching tuple is identified. In year 2020, assume that only one movie was released due to pandemic.

a. In the *best possible scenario*, how long does it take to retrieve the tuple that satisfies the above SQL statement from the disk? Briefly explain how you arrived at your answer. (5 points)

ANSWER:
0.01ms. In the most optimistic scenario, the tuple is stored in the first block of the table, and the disk head happens to be right above this first block. In that case, we just have to read the block from the disk without waiting for the seek time or rotational delay. Since the disk's RPM is 6000, one disk rotation takes 10ms. Since each track has 1,000 sectors, reading one sector from the disk takes 10ms/1,000 = 0.01ms.

b. In the *average case scenario*, how long does it take to retrieve the tuple that satisfies the above SQL statement from the disk? Briefly explain how you arrived at your answer. (5 points)

ANSWER:
16.36ms. In the average case scenario, we have to wait for the average seek time and the average rotational delay, 10ms (average seek time) + 5ms (average rotational delay) = 15ms. In the average case, the tuple that we look for is located in the middle, so we will have to read half of the blocks. Since there are 271 blocks, the middle block is 136. Reading 136 blocks sequentially takes 0.01ms*136 = 1.36ms. So in total, it takes 10ms + 5ms + 1.36ms = 16.36ms

3. For this subproblem, assume that the table is stored as a sorted file in increasing order of (studio, year) attributes. That is, the tuples are first sorted by the studio attribute and then by the year attribute if two tuples have the same studio value. For each of the following statement, circle either True or False and explain your answer briefly. Assume a single-level index unless indicated otherwise. (5 points)

   a. It is possible to build a sparse index on title. TRUE / FALSE

      ANSWER:
      False. A sparse index can be built only if all tuples in the table have been sorted by the search-key attribute of the index.

   b. It is possible to build a dense index on studio. TRUE / FALSE

      ANSWER:
      True. We can build a dense index on any attribute.

   c. It is possible to build a sparse index on year. TRUE / FALSE

      ANSWER:
      False. A sparse index can be built only if all tuples in the table have been sorted by the search-key attribute of the index.

   d. It is possible to build a dense index on year. TRUE / FALSE

      ANSWER:
      True. We can build a dense index on any attribute.

   e. It is possible to build a sparse index on studio. TRUE / FALSE

      ANSWER:
      True. All tuples in the table appear in the lexicographical order of studio.

Problem 3: Index (20 points)

1. Consider the Movie table with the following schema:

   Movie(title, year, studio, prequel, revenue)

   The revenue column has the box-office revenue for the movie. The prequel column has the title of the movie's prequel if any and NULL otherwise. Assume that a movie's title is unique even though our table definition does not include either the primary key or the unique constraint on title.

   Consider the following two queries that we want to execute repeatedly on the table

```
Q1: SELECT SUM(revenue) FROM Movie WHERE studio={studio} GROUP BY year;
Q2: SELECT M1.title
    FROM Movie M1, Movie M2
    WHERE M1.prequel = M2.title AND M1.revenue > M2.revenue;
```

   Note that `{studio}` in Q1 is provided by the user when the query is executed so this value varies in each execution.

   a. Assume that we want to build *two* indexes to support the previous two queries efficiently. Specify two attributes, one attribute per each index on which the index should be built. (4 points)

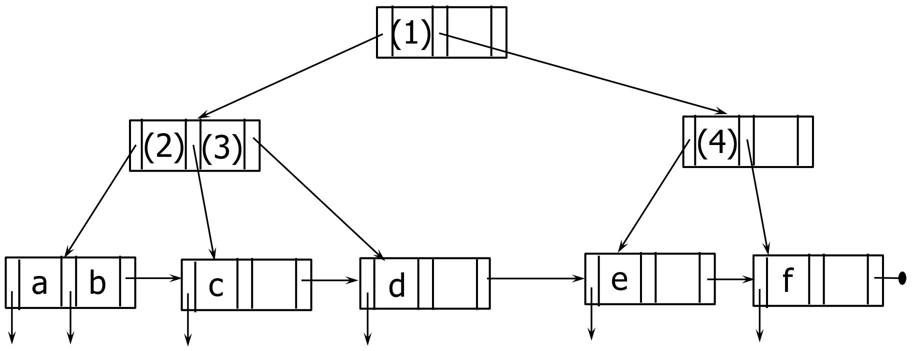
   ANSWER:
   studio and (title or prequel). Q1 will speed up significantly from an index on studio due to the selection condition 'studio=studio'. Q2's join will speed up significantly if we build an index on either title or prequel. Building an index on revenue will also help, but its speedup will be significantly less than an index on either prequel or title

   b. State which of the two indexes should be clustered. Assume that both queries are executed with roughly equal frequencies and we do not run any other query on the table. Explain your answer briefly. (4 points)

   ANSWER:
   For Q1, we retrieve multiple tuples with 'studio=studio' so it is worthwhile to cluster tuples with the same studio values to reduce the number of blocks to be read to retrieve those tuples. For title or prequel, there is only one tuple with the given condition, so whether we cluster them or not does not affect our performance.

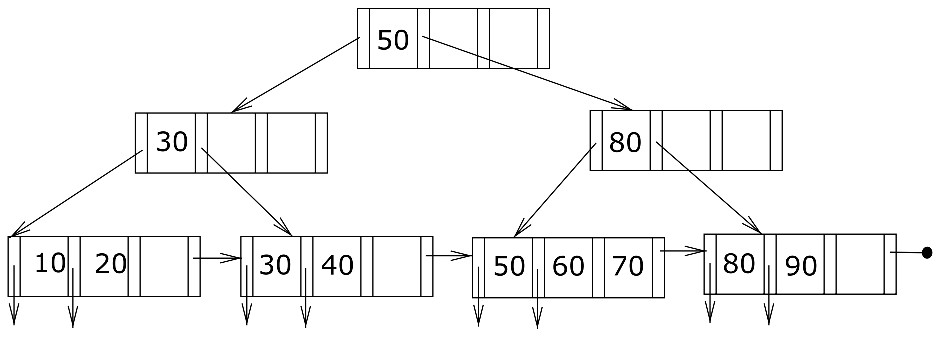
2. Consider the B+Tree insertion algorithm that we learned in the class. Assume that if there is an overflow after an insertion and a node is split, the left node gets 1 more keys/pointers than the right node, if the overflowing keys/pointers cannot be evenly distributed between the two nodes.

   Consider the following B+Tree constructed using the algorithm by inserting the keys a, b, ..., f in the reverse order. In the space provided below, write down the key values that should be at the positions labeled as (1), (2), (3), and (4) in the tree. (4 points)



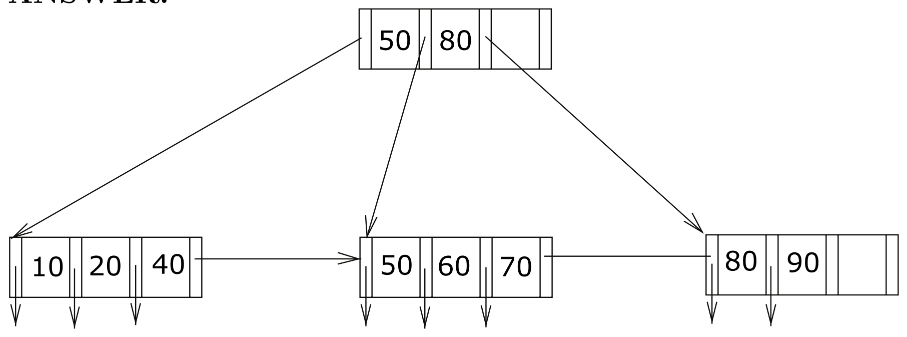
ANSWER:
(1) e (2) c (3) d (4) f

3. Here is another B+tree.



Suppose that we delete the tuple with key 30 from this tree. Draw the new tree after the deletion by using the B+Tree deletion algorithm learned in the class. For partial credit, it may be a good idea to write down every intermediate state of the tree during the deletion. (8 points)

ANSWER:



Problem 4: Cost of Join (20 points)

Suppose we have 2 tables, $R(A, B)$ and $S(B, C)$, with the following characteristics:

- $|R| = 5,000$ (number of tuples of R), $b_R = 500$ (number of blocks of R).
- $|S| = 10,000$ (number of tuples of S), $b_S = 1,000$ (number of blocks of S).
- Neither table is sorted by any attribute.
- For every tuple of R, there is exactly one matching tuple in S with the same B value.
- We have a dense B+Tree of $n = 50$ constructed on the $S.B$ attribute. The B+Tree is of height 3 with 1 root node, 10 nonleaf nonroot nodes and 200 leaf nodes. Each node of the B+Tree corresponds to one disk block.
- We have 10 blocks of memory buffer that can be used for query processing.

Consider the following SQL statement:

```
SELECT * FROM R, S WHERE R.B = S.B
```

We decided to execute the above query by performing an index join using the B+Tree on $S.B$. In solving this problem, assume the cost model that we learned in the class. That is, the cost model counts the number of disk blocks read/written during the join, excluding the cost of writing the final result of the join.

1. In the space provided below, provide a bullet list of how you will to use the 10 main memory blocks to perform the index join as efficiently as possible. If you use parts of the memory to "cache" the table and/or index, be specific about what blocks/nodes you will cache. For example, your answer may look like the following:

    - 3 memory blocks to read blocks from R sequentially
    - 3 memory blocks to read and cache blocks of S with the matching tuples
    - 1 memory blocks to buffer the output tuples
    - 3 memory blocks to read and cache B+Tree nodes, one memory block per each level of the tree.

    Briefly explain answer. (10 points)

    ANSWER:
    Use 1 memory block to read R blocks sequentially. Use 1 memory block to buffer the output tuples. Use 1 block to cache the B+Tree root node, 6 blocks to cache 6 nonleaf nonroot nodes, and 1 block to read the leaf node and then to read the S block that has the matching tuple.

    If your solution is to use 5 blocks to cache 5 nonleaf nonroot nodes, 1 block to read and cache the leaf node, and 1 block to read and cache the matching S blocks, we still gave the full credit.

2. Under the efficient memory-usage policy, what is the expected index-traversal cost per each tuple of R, excluding the initial caching cost (i.e., the number of disk blocks that need to be read to traverse the index to identify the location of the matching S tuple given an R tuple)? (5 points)

   ANSWER:
   1.4. The root node is always cached, so there is no traversal cost. Since we cache 6 out of 10 nonleaf nonroot nodes, the expected cost for this level is 0.4. Since we cache none of the leaf nodes, the expected cost for the leaf level is 1. So the expected cost is 1.4.

   If your answer was to use 5 blocks to cache 5 nonleaf nonroot nodes, and 1 block to read the leaf node, the expected cost is $0.5 + 0.995 = 1.495$.

3. If you want to reduce the expected index-traversal cost per every tuple of R to be 0.5 or less (excluding the initial caching cost), what is the minimum number of main memory blocks that you need? Briefly explain your answer (5 points)

   ANSWER:
   We need to cache the root node, all nonleaf nonroot nodes, and half of the leaf nodes to reduce the cost to less than 0.5. So we need to have 1+10+100 memory buffer to cache the index nodes. Including other requirements (1 for R, 1 for S and 1 for output), we need at least 114 main memory blocks.

Problem 5: NoSQL (10 points)

Consider the table $R(A, B, C)$. All columns of the table are integers. The table has no NULL values.

Assume that the text file `R.txt` contains the same data as R. That is, each line of `R.txt` corresponds to a tuple in R with the column values of A, B, and C appearing in this sequence separated by a space character.

Now someone wrote the following PySpark code to analyze the data in `R.txt`:

```
a = sc.textFile("R.txt")
b = a.map(lambda x: x.split(" "))
c = b.filter(lambda x: int(x[0])>10)
d = c.map(lambda x: (int(x[0]), 1 if int(x[1]) < 50 else 2, int(x[2])))
e = d.map(lambda x: ((x[0], x[1]), x[2]))
f = e.reduceByKey(lambda a, b: a+b)
g = f.filter(lambda x: x[1] > 20)
h = g.map(lambda x: (1, x[0][0]))
i = h.reduceByKey(lambda a, b: a if (a < b) else b)
j = i.map(lambda x: x[1])
j.saveAsTextFile("output")
```

You may assume that the table R and the file `R.txt` has enough tuples to output a non-empty result from the above code. The main question for this problem is given on the next page.

In the space provided below, write a SQL query that returns an equivalent answer to the above PySpark code based on the table *R*. Write your query succinctly and neatly. You may get as few as 0 point if your solution is far more complicated than necessary, or if we cannot understand your solution. Note that your SQL query just needs to return an equivalent answer. It does *not* have to save the result in the `output` file/directory as the PySpark code does.

ANSWER:

```
WITH S AS
    (SELECT A
     FROM R
     WHERE A > 10
     GROUP BY A, CASE WHEN (B < 50) THEN 1 ELSE 2 END
     HAVING SUM(C) > 20)
SELECT MIN(A) FROM S;
```

Problem 6: Transactions (10 points)

Consider the relation Movie(id, title, studio, budget), and the following transaction T:

```
(Q1) SELECT SUM(budget) FROM Movie WHERE studio = 'Disney';
    <other SELECT statements that only READ data from the database>
(Q2) SELECT SUM(budget) FROM Movie WHERE studio = 'Disney';
COMMIT
```

1. Suppose all other transactions in the system are declared as SERIALIZABLE, and they involve only SELECT statements and an UPDATE of an existing tuple on the budget attribute. What is the weakest isolation level needed for transaction T to ensure that queries Q1 and Q2 will always get the same result? Circle one and briefly explain your answer:

   1. READ UNCOMMITTED
   2. READ COMMITTED
   3. REPEATABLE READ
   4. SERIALIZABLE

   ANSWER:
   REPEATABLE READ. Since there is no insertion, we do not have to worry about phantoms, but we do need to worry about non-repeatable reads

2. Suppose all other transactions in the system are declared as SERIALIZABLE, and we know nothing else about them. What is the weakest isolation level needed for transaction T to ensure that queries Q1 and Q2 will always get the same result? Circle one and briefly explain your answer:

   1. READ UNCOMMITTED
   2. READ COMMITTED
   3. REPEATABLE READ
   4. SERIALIZABLE

   ANSWER:
   SERIALIZABLE. A new tuple may be inserted and create the phantom problem. SERIALIZABLE is the only isolation level that prevents phantoms