CS 245 Midterm Exam
Winter 2020

- Please read all instructions (including these) carefully.
- There are 4 problems, some with multiple parts, for a total of 60 points. You have 80 minutes to work on the exam.
- The exam is open notes. You may use a laptop with all communication facilities (Wi-Fi, Bluetooth, etc) disabled.
- Please write your answers in the space provided on the exam, and clearly mark your solutions. You may use the backs of the exam pages as scratch paper. Please do not use any additional scratch paper.
- Solutions will be graded on correctness and clarity. For the long-answer problems, please show your intermediate work. Each problem has a relatively simple to explain solution, and we may deduct points if your solution is much more complex than necessary. Partial solutions will be graded for partial credit.

NAME: _____

SUID: _____

In accordance with both the letter and spirit of the Honor Code, I have neither given nor received assistance on this examination.

SIGNATURE: _____

Grading: P1=20 points, P2=12 points, P3=10 points, P4=18 points, total: 60 points.

Problem 1: Short Answer Questions (20 points)

1. (2 points) Which of the following features of modern relational database systems were available in System R? *(Check all that apply.)*

- ☐ Cost-based optimization.
- ☐ Transactions.
- ☐ Indexes.
- ☐ Multiple isolation levels.

2. (2 points) System R used SQL views for access control (defining what data each user is allowed to see). Which of the following data access policies **cannot** be specified using view-based access control? *(Circle your answer.)*

(a) User Alex can only query the Orders table.

(b) User Bilal can only query records in the Orders table where the "country" field is USA.

(c) User Chen can only query the database from the office network.

(d) User Dana can only see the sum of Orders placed each day, but not individual orders.

3. (2 points) Consider a table People with attributes SALARY and NAME. Given the relational algebra expression $\Pi_{NAME}(\sigma_{SALARY>10,000}(\Pi_{NAME \cup SALARY}(People))$, which of the following are valid rewrites? *(Check all that apply.)*

- ☐ $\Pi_{NAME}(\sigma_{SALARY>10,000}(People))$
- ☐ $\Pi_{NAME}(\sigma_{SALARY>10,000}(\Pi_{SALARY}(People))$
- ☐ $\sigma_{SALARY>10,000}(\Pi_{NAME \cup SALARY}(People)$
- ☐ $\sigma_{SALARY>10,000}(\Pi_{NAME}(\Pi_{NAME \cup SALARY}(People))$
- ☐ $\Pi_{NAME}(\sigma_{SALARY>10,000}(\Pi_{NAME}(\Pi_{SALARY}(People)))$

4. (2 points) Suppose that you have access to hard disks that can read or write data at 100 MB/s. You want to combine several of these disks into a RAID configuration. What are the maximum read and write throughputs that you can achieve under each of the following configurations?

(a) 2 disks using RAID 0: Read = __200___ MB/s, Write = __200___ MB/s

(b) 2 disks using RAID 1: Read = __200___ MB/s, Write = __100___ MB/s

(c) 4 disks using RAID 5: Read = __400___ MB/s, Write = __300___ MB/s

5. (2 points) Modern data management systems typically focus on either transactional or analytical workloads. Which type of workload is each of the following features **most likely to improve performance** in? *(Circle one answer for each row.)*

(a) Vectorization:	TRANSACTIONAL	ANALYTICAL
(b) Fine-grained locking:	TRANSACTIONAL	ANALYTICAL
(c) RLE compression:	TRANSACTIONAL	ANALYTICAL
(d) Z-ordering:	TRANSACTIONAL	ANALYTICAL

6. (2 points). What is the **logical data model** in PyTorch, and what are two different **physical storage formats** that can be used to store a data item in PyTorch?

Logical: __Tensors_____

Physical 1: __Storing a matrix in row-major order_____

Physical 2: __Storing a matrix in column-major order_____

Note: other answers were okay too, e.g. sparse vs dense storage, floating-point precision, etc.

7. (2 points). Suppose that we want to join a table R with fields a_1, …, a_n against n other tables S_1, …, S_n, where each attribute a_i in R matches a key in the corresponding table S_i. This is called a "star join". How many valid **left-deep join plans** exist for this query? *(Circle your answer.)*

(a) $\Theta(n)$ (b) $\Theta(2^n)$ (c) $\Theta(n!)$ (d) $\Theta(n^n)$

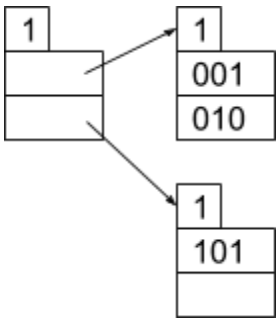
8. (2 points) Spark SQL allows Data Source plugins to perform filters and projections based on their knowledge of the data format being queried. For each of the storage formats below, specify whether knowing a query's filters or projections can reduce the **number of I/Os** to answer it. *(Assume each record is smaller than 1 disk block. Check the relevant boxes for each row.)*

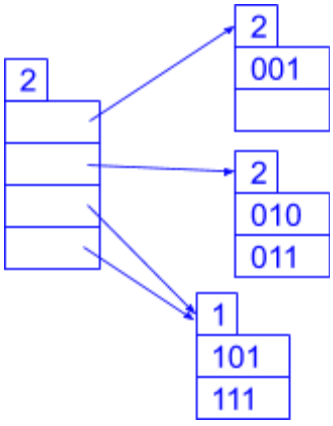
Data format	**Filters help**	**Projections help**
Row-oriented table with variable-format records		
Row-oriented table with fixed-format records		
Column-oriented table partitioned by the "date" field	✓	✓
Column-oriented, unsorted table	✓	✓

9. (2 points) C-store designed many of its compression formats to be byte-aligned: for example, with null suppression, a 32-bit integer would be stored as 1, 2, 3, 4 or 5 bytes. Why did the authors choose to make their formats byte-aligned?

Allowing values to be stored in a fractional number of bytes would create more CPU work for compression and decompression, which the authors said was not worth it.

10. (2 points) Consider the extendible hash table below, with space for 2 keys per bucket and a hash function that returns 3 bits per key. Each cell shows the hash values of keys that are in it. Draw how the table will look after we insert keys with hashes 011 and 111, in that order.





Problem 2: Storage Hardware (12 points)

a) **(2 points)** You have 10,000,000 rows of data in a row store on a hard drive. Each row fits exactly in one disk block, and the blocks for the table are sequential. The hard drive has a 10ms seek latency to start a read operation, and after it starts, it can read 200,000 blocks per second. How much time will you spend doing I/O to read the entire table?

Since the data is contiguous, you seek once, then read all of it.
10ms + 10,000,000 blocks / (200,000 blocks/second) = 0.01 + 50

50.01 seconds

b) **(2 points)** You now add a B+ tree index on column 0. Column 0 has unique values. Each node in the B+ tree fits in exactly one block, but the blocks of the tree are not contiguous. The tree has 3 layers, and no nodes are cached in memory. Suppose that you now want to read rows for a random set of values of column 0 using this tree. Assume the column 0 values are random, so they each require another traversal of the B+ tree. How long will you spend doing I/O to retrieve R rows from the table using the index, as a function of R?

For each record you need, you must read 3 blocks in the index, and one in the data table. All the blocks are scattered, so it is 4 total seeks. You read 4R total blocks, and each requires one seek plus transfer time.

4R (0.01 + 1/200,000)

0.04002R seconds

c) **(2 points)** You can perform this query by either scanning over all the rows directly, checking the value of column 0 in each, or by traversing the B+ tree to find the relevant rows. For what range of selectivities is it faster to use the index (I/O time only)?

We compare the runtimes to get the breakeven point.
0.04002R < 50.01
R < 0.0125%

It is fastest to use the index if the selectivity is LESS / GREATER (circle one) than

0.0125%

d) **(2 points)** Now you replace your hard drive with an NVMe SSD that takes 0.02ms to begin a read operation, and can read 4,000,000 blocks per second once it begins reading. How long will you spend doing I/O for a direct table scan?

Same calculations as for a hard drive. 1 seek plus transfer time.
0.00002 seconds + (10,000,000 blocks)/(4,000,000 blocks/second)

2.50002 seconds

e) **(2 points)** How long will you spend doing I/O to retrieve R rows from the table predicated on column 0, as a function of R? Make all the same assumptions as in b).

For each record you need, you must read 3 blocks in the index, and one in the data table. All the blocks are scattered, so it is 4 total seeks. You read 4R total blocks, and each requires one seek plus transfer time.

4R (0.00002 + 1/ 4,000,000)

0.000081 * R seconds

f) **(2 points)** For what range of selectivities is it fastest to use the index on the NVMe SSD?

We compare the runtimes to get the breakeven point.
0.000081R < 2.50002
R < 0.3%

It is fastest to use the index if the selectivity is LESS / GREATER (circle one) than

about 0.3%

Problem 3: Database System Architectures (10 points)

In this question, we compare the performance of different DBMS architectures. Suppose we have a table of people, their birthdays, and their states of residence. Rows might look like this:

Person	**Birthday (yyyy-mm-dd)**	**State**
Shivaram	1955-02-13	NC
Emma	2005-08-30	CA

We want to make two queries on this table:

1. SELECT * FROM table WHERE 1993-01-01 ≤ Birthday AND Birthday ≤ 1993-12-31
2. SELECT Person FROM table WHERE State=CA

Assume the table can be laid out in two different formats:

- A row store indexed by B+ trees on the Birthday and State columns.
- A column store with bitmap indexes like in the C-store paper on the Birthday and State columns: For each unique value in the indexed column, we store a bit vector of the same length as the column with a 1 in all rows containing that value and a 0 in all other rows.

Assume there are *N* people with *M* unique birthdays who all live in the fifty states.

a) (2 points) Assume we execute Query 1 on the row store by finding the list of matching tuples using the B+ tree index and then reading them from the table. What is the time complexity of this query plan? *Briefly explain your answer.*

(a) Θ(N) (b) Θ(N log M) (c) Θ(N + log M) (d) Θ(M + log N)

b) (2 points) Assume we execute Query 1 on the column store by OR-ing the bitmap indexes for each date in the range together and retrieving the marked tuples from the table. What is the time complexity of this query plan, assuming at least one person in the table was born on every day in 1993? *Briefly explain your answer.*

(a) Θ(N) (b) Θ(N log M) (c) Θ(N + log N) (d) Θ(M + log N)

c) **(3 points)** Assume that N=1,000,000, M=30,000, and the data and indexes are stored on hard disks. If the distribution of birthdays is uniform over the past eighty years, which storage layout is likely to perform best **in practice** on Query 1? *Please explain your reasoning and any assumptions you make.*

The row store with B+ trees because the lists of tuples associated with consecutive keys are stored consecutively in the tree, minimizing the number of disk loads/memory accesses made. By contrast, with a column store, determining which tuples to return would require reading hundreds of bitmaps or doing a full column scan. Moreover, because we return entire rows, retrieving tuples from individual columns is not efficient.

d) **(3 points)** Assume that N=1,000,000 and the data is stored on hard disks but the indexes are stored in memory. If 300,000 of the people live in California, which storage layout is likely to perform best **in practice** on Query 2? *Please explain your reasoning and any assumptions you make.*

The column store would perform the fastest because the database can scan the Person column quickly, using the bitmap index to know which entries to return. Scanning the row store would require loading full rows, which is inefficient.

Problem 4: Cost-Based Optimization (18 points)

In this problem, we will do cost-based analysis and optimize query execution.

a) **(4 points)** Consider query $\sigma_{A<15\ AND\ C\geq10}(X \bowtie Y)$ on relations X(A, B) and Y(B, C). *(Recall that the notation X(A, B) means that relation X has attributes A and B.)* We present collected statistics about our relations below, where T(R) is the number of tuples in a relation and DOM(R, A) is the domain of values of attribute A in a relation (Note: data values are **uniformly and independently distributed** in all columns):

T(X) = 3000 T(Y) = 2000
DOM(X, A) = 30 (integers from 0 inclusive to 30 exclusive)
DOM(X, B) = 20 (integers from 0 inclusive to 20 exclusive)
DOM(Y, B) = 10 (integers from 5 inclusive to 15 exclusive)
DOM(Y, C) = 20 (integers from 0 inclusive to 20 exclusive)

Firstly, ignore the selection operator in the query, and estimate the following statistics:

$T(X \bowtie Y)$ = _300,000__

$DOM(X \bowtie Y, A)$ = __30___ (integers from __0_ inclusive to _30_ exclusive)
$DOM(X \bowtie Y, B)$ = __10___ (integers from __5_ inclusive to _15_ exclusive)
$DOM(X \bowtie Y, C)$ = __20___ (integers from __0_ inclusive to _20_ exclusive)

b) **(8 points)** Now consider the complete query, $\sigma_{A<15\ AND\ C\geq10}(X \bowtie Y)$. The first optimization we can make to this query is to push down selections, obtaining an expression of the form $R_1 \bowtie R_2$. What are the expressions R_1 and R_2 we get after pushing down selections?

R_1 = ____$\sigma_{A<15}(X)$_____ R_2 = _____$\sigma_{C\geq10}(Y)$_____

Next, compute the following statistics estimates:

$T(R_1)$ = __1500__ $T(R_2)$ = __1000__ $T(R_1 \bowtie R_2)$ = _75,000____

$DOM(R_1 \bowtie R_2, A)$ = __15___ (integers from _0__ inclusive to _15_ exclusive)
$DOM(R_1 \bowtie R_2, B)$ = __10___ (integers from _5__ inclusive to _15_ exclusive)
$DOM(R_1 \bowtie R_2, C)$ = __10___ (integers from _10_ inclusive to _20_ exclusive)

c) **(6 points).** Following part b), we then estimate the I/O cost of executing the plan $R_1 \bowtie R_2$. Suppose the sizes of R_1 and R_2 are **150** and **100** blocks, respectively. For the questions below, **ignore the I/O for writing the final join result back to disk.** If a table needs to be sorted during a join, we use the 2-pass merge sort discussed in the lecture and write sorted table back to disk that takes a total of 4·b of I/Os for a table containing b blocks. Please compute the number of I/Os (blocks read and written) needed to join the two tables with each of the strategies below *(make sure to explain your work)*:

Nested loop join: Every time you read one block of R_1 into memory, build an in-memory hash table for the tuples in that block, and loop through R_2 to find tuples with the same B column.

B(R1) + B(R1)*B(R2)
150 I/Os to read R1
100 I/Os of R2 for each block of R1
150 + 150 * 100 = 15,150

Total I/Os: ___15,150___

Merge join: The two input tables must be sorted by the join attribute to produce an output table that also goes to disk and then execute the merge sort. Assume R_1 is sorted on A, while R_2 is sorted on B.

4B(R1) + B(R1) + B(R2)
4 * 150 to sort
150 + 100 to merge
4 * 150 + 150 + 100 = 850

Total I/Os: ____850____

Hash join: Suppose there are 21 buffers in memory. Each buffer can hold one block of data. To do hash join, each relation is first hashed to 20 buckets and written back to disk. Then for each pair of buckets, the smaller one is loaded into memory and joined with each tuple in the larger one. (You may assume the smaller bucket can fit in memory.)

3(B(R1) + B(R2))
B(R1) + B(R2) blocks to read into memory and hash into buckets
B(R1) + B(R2) blocks to write all buckets back
Read in each pair of buckets and join: B(R1) + B(R2)
3 * (150 + 100) = 750

Total I/Os: ____750_____