Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.830 Database Systems: Fall 2008 Quiz II

There are 14 questions and 11 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.
NO PHONES, NO LAPTOPS, NO PDAS, ETC.
YOU MAY USE A CALCULATOR.

Do not write in the boxes below

1-3 (xx/14)	4-6 (xx/26)	7-8 (xx/10)	9-11 (xx/26)	12-14 (xx/24)	Total (xx/100)

Name: Solutions

I Short Answer

1. [4 points]: List two reasons why a column-oriented design is advantageous in a data warehouse environment.

(Write your answer in the spaces below.)

A. *Data warehouse workloads involve large scans of a few columns, meaning that column-stores read less data from disk*

B. *Columns tend to compress better than rows, which is good in a read intensive setting.*

Other answers are possible.

2. [4 points]:

Which of the following statements about the MapReduce system are true:

(Circle T or F for each statement.)

(T) F A MapReduce job will complete even if one of the worker nodes fails.

Worker tasks are restarted at other nodes when a failure occurs.

T (F) A MapReduce job will complete even if the master node fails.

There is only a single master node, so if it fails the entire MapReduce job must be restarted.

T (F) Map workers send their results directly to reduce workers.

Map workers write their results to their local disk. The reducers pull from the disks when needed. Due to ambiguity about if this is "sending directly" or not, this question was excluded.

(T) F If there are M map tasks, using more than M map workers may improve MapReduce performance.

MapReduce will start multiple copies of the last few map or reduce tasks, which can avoid issues with slow nodes.

3. [6 points]:

Which of the following statements about the two-phase commit (2PC) protocol are true:

(Circle T or F for each statement.)

(T) F In the "no information" case (i.e., when a subordinate asks the coordinator about the outcome of a transaction that the coordinator has no information about), the coordinator running the presumed abort version of 2PC returns the same response to a subordinate as it would in basic 2PC.

Both versions answer "ABORT."

T (F) The Presumed Commit protocol reduces the number of forced writes required by the coordinator compared to basic 2PC for transactions that successfully commit.

Presumed commit forces a COLLECTING record to disk (and on the root coordinator, a COMMIT record), while the traditional protocol forces a COMMIT record.

(T) F The Presumed Abort protocol reduces the number of forced writes required of the coordinator compared to basic 2PC for transactions that abort.

With presumed abort the coordinator does not need to force write any entries in the abort case.

(T) F In basic 2PC, a subordinate that votes to abort a transaction T receives no more messages pertaining to T, assuming neither T's coordinator nor any of T's subordinate nodes fail.

The coordinator does not need to alert the aborted subordinate about the abort.

II C-Store

Dana Bass is trying to understand the performance difference between row- and column- oriented databases. Her computer's disk can perform sequential I/O at 20 MB/sec, and takes 10 ms per seek; it has 10 MB of memory. She is performing range queries on a table with 10 4-byte integer columns, and 100 million rows. Data is not compressed in either system, and all columns are sorted in the same order. Assume each column is laid out completely sequentially on disk, as is the row-oriented table.

4. [16 points]: Fill in the table below with your estimates of the *minimal* I/O time for each operation, in a row-store and a column-store. Assume that all reads are on contiguous groups of rows, and that queries output row-oriented tuples.

Solution:

The cost for scanning sequential rows in a row-store is a seek to the first row, then a sequential scan of the remaining rows. We assume that the data can be processed as it is scanned - one thread sequentially loads tuples from disk while another applies any predicates or materialization in parallel. In this scenario, having 10MB of RAM is the same as having 1GB or 1MB of RAM.

In a column-store, for each extra column one reads in parallel, the disk head must seek to a different file containing that column at a given offset. Thus, the best strategy is to read in $\frac{10}{columns\_read}$ *MB of each column in to memory (incurring one seek for each such read), materialize the row, process it in memory, and move to the next sequential chunk of columns. In the special case where you only read one column, treat the behavior of the datbase as you would in a row-store with one column.*

No. Columns Read	*No. Rows Read*	*Column-Oriented System*	*Row-Oriented System*
1	*10 Million*	$\frac{10Mrows*4bytes*1column}{20MB/sec} = 2sec$ *+ 10 ms seek* *2.1 sec total*	$\frac{10Mrows*4bytes*10columns}{20MB/sec} = 20sec$ *+ 10 ms seek* *20.1 sec total*
10	*10 Million*	*20 sec sequential read (400MB of data)* *10MB RAM → 1 MB/column* *read 400 MB in 1 MB chunks* *400 seeks * 10 msec = 4 sec* *24 sec total*	*20 sec, 10 ms seek* *20.1 sec total* *(same as above)*

III BigTable

Consider an example table storing bank accounts in Google's BigTable system:

Key	Value
"adam"	\$300
"evan"	\$600
"sam"	\$9000

The bank programmer implements the deposit function as follows:

```
function deposit(account, amount):
  currentBalance = bigtable.get(account)
  currentBalance += amount
  bigtable.put(account, currentBalance)
```

5. [4 points]: Given the table above, a user executes deposit("sam", \$500). The program crashes somewhere during execution. What are the possible states of the "sam" row in the table? Provide a short explanation for your answer.

(Write your answer in the space below.)

The row could either have "sam" = \$9000 or "sam" = \$9500, depending on if the program crashed before or after it wrote the result.

6. [6 points]: Given the table above, a user executes deposit("sam", \$500). The program completes. Immediately after the program completes, the power goes out in the BigTable data center, and all the machines reboot. When BigTable recovers, what are the possible states of the "sam" row in the table? Provide a short explanation for your answer.

(Write your answer in the space below.)

The row can only have the value \$9500, since BigTable does not inform a client that a write has been completed until it logs the update to disk on multiple physical machines. Thus, when BigTable recovers it will find the write in the log.

Name:

Now consider the following pair of functions:

```
function transfer(sourceAccount, destinationAccount, amount):
  sourceBalance = bigtable.get(sourceAccount)
  if sourceBalance < amount:
     raise Exception("not enough money to transfer")
  sourceBalance −= amount

  destinationBalance = bigtable.get(destinationAccount)
  destinationBalance += amount

  bigtable.put(sourceAccount, sourceBalance)
  bigtable.put(destinationAccount, destinationBalance)
```

```
function totalAssets():
  assets = 0
  for account, balance in bigtable:
     assets += balance
  return assets
```

7. [6 points]: Given the table above, a program executes transfer("evan", "adam", 200). While that function is executing, another program executes totalAssets(). What return values from totalAssets() are possible? For the purposes of this question, assume there is only one version of each value, and that the values in BigTable are stored and processed in the order given above (i.e., "adam", "evan", "sam"). Provide a short explanation for your answer.

(Write your answer in the space below.)

$9,900 is possible if totalAssets either reads all the balances before or after the transfer modifies any of them. $9,700 is possible if totalAssets reads "adam" before the $200 is added, but reads "evan" after $200 is deducted.

It might seem like $10,100 is possible, by reading "adam" after adding $200, and reading "evan" before the subtract. However, this is not possible. The transfer function performs the subtract, then the add. The scan in totalAssets reads the rows in order. Thus, if it sees the modified version of "adam" it will see the modified version of "evan." However, it would not be a good idea to write an application that depends on this behavour, as it depends on the order of the keys and the operations in the code, which could easily change.

8. [4 points]: If the data were stored in a traditional relational database system using strict two-phase locking, and transfer() and totalAssets() were each implemented as single transactions, what would the possible results of totalAssets() be? Provide a short explanation for your answer.

(Write your answer in the space below.)

$9,900 is the only possible solution because the DBMS will use concurrency control to execute the totalAssets transaction either strictly before or after the transfer transaction.

IV Distributed Query Execution

Suppose you have a database with two tables with the following schema:

```
T1 : (A int PRIMARY KEY, B int, C varchar)
T2 : (D int REFERENCES T1.A, E varchar)
```

And suppose you are running the following query:

```
SELECT T1.A,COUNT(*)
FROM T1,T2
AND T1.A = T2.D
AND T1.B > n
GROUP BY T1.A
```

Your system has the following configuration:

Parameter	Description	Value
$\lvert T1\rvert$	Size of table `T1`	300 MB
$\lvert T2\rvert$	Size of table `T2`	600 MB
$\lvert M\rvert$	Size of memory of a single node	100 MB
$\lVert T1\rVert$	Number of records in $T1$	10 million
$\lvert int\rvert$	Size of integer, in bytes	4
T	Disk throughput, in bytes / sec	20 MB/sec
N	Network transmit rate, in bytes / sec	10 MB/sec
f	Fraction of tuples selected by `T1.B > n` predicate	0.5

Name:

9. [9 points]: Suppose you run the query on a single node. Assuming there are no indices, and the tables are not sorted, indicate what join algorithm (of the algorithms we have studied in class) would be most efficient, and estimate the time to process the query, ignoring disk seeks and CPU costs. Include a brief justification or calculation.

(Write your answers in the spaces below.)

Join algorithm:

Read all of A (300MB), applying the predicate and projecting to keep only T1.A. Build an in-memory hash table with the results: 10 million tuples in T1 $\times \frac{1}{2}$ *= 5 million output tuples* $\times$ *4 bytes per tuple = 20 MB of data. Sequentially scan T2 (600 MB), probing T2.D in the hash table. Emit T1.A for each matching join tuple, and build a hash aggregate to count the number of times T1.A appears (5 million output tuples* $\times$ *(4 bytes id + 4 bytes count) = 40 MB). Finally iterate over the aggregate and output the results to the user.*

Estimated time, in seconds:

Read T1 in $\frac{300MB}{20MB/s} = 15s$*. Read T2 in* $\frac{600MB}{20MB/s} = 30s$*.*

Total time: 45 seconds.

10. [10 points]: Now suppose you switch to a cluster of 5 machines. If the above query is the only query running in the system, describe a partitioning for the tables and join algorithm that will provide the best performance, and estimate the total processing time (again, ignoring disk seeks and CPU cost). You may assume about the same number of `T2.D` values join with each `T1.A` value. Include a brief justification or calculation. You can assume the coordinator can receive an aggregate 50 MB/sec over the network from the five workers transmitting simultaneously at 10 MB/sec.

(Write your answers in the spaces below.)

Partitioning strategy:

Hash tables on T1.A/T2.D. Each node will have approximately 60 MB of T1 and 120 MB of T2. 30 MB of T1 will pass the predicate.

Join algorithm:

Same as above, but with smaller result set on each machine. Final aggregation sent to coordinator. Since tables are properly partitioned, no networking is needed to process joins.

Estimated time, in seconds:

Each node has to read 180 MB from disk, which takes 9 seconds. Each node has 2 million T1 records on it, which are filtered to 1 million by the predicate (4 MB in-memory hash table for probing 4 byte integers). The aggregation takes 8 MB in RAM, since each integer also needs a 4-byte counter. Once the local aggregation is complete, the results are sent over the network to the coordinator in parallel with the other nodes at 10 MB/sec, taking 0.8 seconds to send over the network.

Total time: 9.8 seconds.

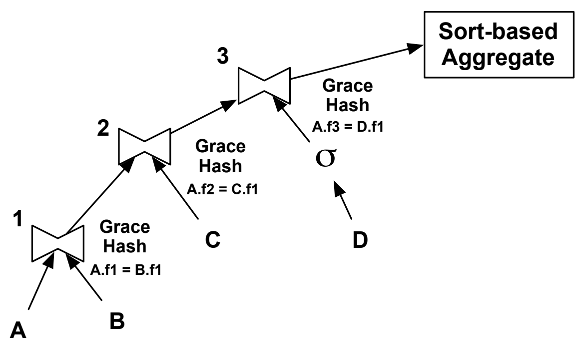
Name:

V Adaptive Query Processing

Jenny Join, after hearing the lecture about adaptive query processing, decides to add several types of adaptivity to her database system.

First, Jenny decides to add simple plan-based adaptivity in the style of the Kabra and DeWitt paper mentioned in the "Adaptive Query Processing Through the Looking Glass" paper and discussed in class.

Jenny runs the following query:



11. [6 points]: To help Jenny understand the Kabra and DeWitt approach, describe one way in which it might adapt to each of the following issues that arise during execution of the query shown above:
(Write your answer in the space below each issue.)

A. After running Join 3, the system observes that many fewer tuples will be input to the sort-based aggregate than predicted by the optimizer.

If sufficient memory has been allocated to the aggregate node, it might switch the aggregate to a hash-based aggregate (or in-memory sort).

B. After running Join 1, the system observes that many fewer tuples will be input to Join 2 than predicted by the optimizer.

If sufficient memory has been allocated to the sort-merge join (Join 2), it might switch the join to an in-memory hash join. Switching Join 2 and Join 3 is likely not a good idea as the cardinality of the input to Join 2 has gone down, not up, so it is unlikely that Join 2 will now be more expensive than Join 3.

C. After running Join 1, the optimizer predicts that many more tuples will be output by Join 2 than predicted before Join 1 ran (you may assume other optimizer estimates do not change significantly).

If the cost of Join 2 is now predicted to be greater than Join 3, the system might switch Join 2 and Join 3.

Name:

Now Jenny decides to add even more adaptivity to her system by allowing it to adapt between using secondary (unclustered) indices and sequential scans during query processing.

Her query optimizer uses the following simple approximate cost model to estimate the time to perform a **range scan** of T tuples from a table A containing $|A|$ disk pages, where the disk can read m pages per second and takes s seconds per seek. Assume that a disk page (either from the index or heap file) holds t tuples and that the scan is over a contiguous range of the key attribute of the index.

For sequential scans, each range scan requires one seek to the beginning of the heap file plus a scan of the entire file:

$$C_{seqscan} = s + \frac{|A|}{m}$$

For a secondary (unclustered) B+Tree, each range scan involves reading $\frac{T}{t \times m}$ leaf index pages (assuming the top levels of the index are cached) plus one seek and one page read from the heap file per tuple (assuming no heap file pages are cached).

$$C_{secondary-btreescan} = \frac{T}{t \times m} + T \times (s + \frac{1}{m})$$

Jenny finds that, when there are correlations between the key of an secondary B+Tree and the key of a clustered B+Tree (or the sort order of a table), the performance of the secondary B+Tree is sometimes much better than the cost formula given above.

12. [6 points]: Explain why correlations between secondary and clustered index keys might overestimate the cost of the $C_{secondary-btreescan}$ model given above.

(Write your answer in the space below.)

If there is a correlation between the secondary and clustered index, then consecutive tuples in the leaf pages of the secondary index may be consecutive (or nearly consecutive) in the clustered index. This will means that in reality there won't be seeks or page reads for every tuple, as in the model.

13. [6 points]: Using the parameters given above, give a better approximation of $C_{secondary-btreescan}$ in the presence of correlations.

(Write your answer in the space below.)

If there is a perfect correlation, and assuming the cost model above for scanning the secondary index leaf pages is correct, then the cost will be approximately $\frac{T}{t \times m} + \frac{T}{t} \times (s + \frac{1}{m})$. *We didvide T by t because there are t tuples per page, and we only have to seek and read additional pages from the clustered index every t tuples.*

Name:

Inspired by the idea of adaptive query processing, Jenny wants to devise a technique to exploit her observation about correlations without having to maintain correlation statistics between the clustered attribute and all attributes over which there is an secondary index.

14. [12 points]: Suggest an adaptive query processing technique that changes the plan while it is running to switch between a sequential scan and a (possibly correlated) secondary index, depending on which is better. Your solution should not require precomputing any additional statistics. Be sure to indicate how your approach avoids producing duplicate results when it switches plans.

(Write your answer in the space below.)

Start using secondary indices, assuming correlations are present.

If the number of seeks performed while scanning some small fraction of the range (e.g., the first 1%) is close to what the original secondary model predicts, switch to sequential scan if the overall cost will be lower.

Avoid duplicates by keeping a list of secondary index key values that the system has already scanned, and filtering out tuples in that list. Since the scan is done in order of the secondary index key, this list be efficiently represented as a range of keys (e.g., a (lower bound, upper bound) pair).

End of Quiz II

Name: