University of Waterloo
Midterm Examination Sample Solution
Winter, 2012

1. (4 total marks)

Suppose that a relational database contains the following large relation:

```
Track(ReleaseID, TrackNum, Title, Length, Location)
```

The database has a clustered B+-tree index on the composite key `ReleaseID,TrackNum`. It also has an unclustered B+-Tree index on `TrackNum`.

a. (2 marks) Give an example of a simple, single-table SQL query (not an update, insertion or deletion) that would require fewer I/O operations to answer using the clustered index than the unclustered index. Your example should also require fewer I/O operations to answer using the clustered index than using a full table scan, i.e., than not using either index. For full credit, explain briefly (in one or two sentences) why the query would require less I/O if the clustered index were used.

```
select * from Track where ReleaseID > 'xxx'
```

The secondary index cannot be used to execute this query. It will be faster to execute this query using the clustered index than using a full table scan if the predicate is selective enough, i.e., if enough records do not match the predicate, since the index can be used to avoid reading non-matching tuples.
Another possible answer would be a query that returns results ordered by ReleaseID or by ReleaseID,TrackNum

b. (2 marks) Give an example of a simple, single-table SQL query (not an update, insertion, or deletion) that would require fewer I/O operations to answer using the unclustered index than the clustered index. Your example should also require fewer I/O operations to answer using the clustered index than using a full table scan, i.e., than not using either index. For full credit, explain briefly (in one or two sentences) why the query would require less I/O if the unclustered index were used.

```
select (distinct) TrackNum from Track
```

This query can be answered using only the unclustered index - no access to the data would be required. It could also be answered using the clustered index, but that index will be larger.
Another possible answer would be a query with a predicate on TrackNum, but only if few tuples satisfy the predicate - otherwise a table scan may be faster.

2. (8 total marks)

Suppose that a database system has a hash index on a relation R. The index is implemented using extensible hashing. Suppose that a client inserts a new record into R, and that the insertion causes the size of the hash index directory to double, from n entries to $2n$ entries. For the purposes of this question, assume that there have not been any deletions from this index - only insertions.

a. (2 marks)
Immediately prior to the insertion that doubles the directory size, how many hash buckets did this index have? (Hash buckets are what the directory entries point to.) Give two answers: the maximum number of hash buckets it might have had, and the minimum number of hash buckets it might have had. Express your answers in terms of n.

At a minimum, one new hash bucket will be created each time the directory doubles. When $n = 2$, there will be two hash buckets. When $n = 4$, there will be at least three hash buckets. When $n = 8$ there will be at least four hash buckets, and so on. In general, the minimum number of hash buckets for a directory of size n will be $\log_2 n+1$. The maximum number of hash buckets for a directory of size n is n. In this case, each directory entry points to a distinct hash bucket.

b. (2 marks)
Let k represent the exact number of hash buckets in the index immediately before the insertion that doubles the directory size. How many hash buckets will the index have immediately after the insertion that doubles the directory size? Express your answer in terms of k.

Directory doubling occurs when a hash bucket fills up and it is necessary to split that bucket - resulting in the creation of one new bucket. Immediately after the insertion, the number of buckets will be $k + 1$.

c. (2 marks)
Immediately after the insertion that doubles the directory size, how many hash buckets will have exactly one directory entry pointing to them?

Two buckets (the bucket that split, and the newly-created bucket) will have exactly one directory entry pointing to them. All other buckets will have at least two.

d. (2 marks) Immediately after the insertion that doubles the directory size, what is the maximum number of directory entries that could be pointing at a single hash bucket in the index?

If a hash bucket never splits, it will double its pointer count each time the directory doubles. Thus, as many as half of the directory entries (n out of $2n$) could point at a single bucket in the most extreme case.

3 (8 total marks)

Consider a database that contains the following two relations

```
Track(TrackID,Title,ReleaseID,Duration,Format,Genre)
Release(ReleaseID,Title,ArtistID,ReleaseDate)
```

The DBMS executes the following query:

```
select T.Title,T.Genre,R.ReleaseDate,R.ReleaseID
from Track T, Release R
where T.ReleaseID = R.ReleaseID
and T.Format = 'mp3'
```

a. (4 marks)

Suppose that the DBMS executes this query using a block nested loop join iterator with `Track` as the outer relation and `Release` as the inner. Furthermore, suppose that the access method for `Track` produces `Track` tuples in `Genre` order, and the access method for `Release` produces `Release` tuples in `ReleaseDate` order. Which of the following statements must be true? (Identify all that are true.)

- ~~The output of the join will be in `ReleaseDate` order.~~
- The output of the join will be in `Genre` order. (**TRUE**)
- ~~The output of the join will be in `ReleaseID` order.~~
- ~~The join will consume all `Track` tuples before producing any output tuples.~~
- ~~The join will consume all `Release` tuples before producing any output tuples.~~
- ~~The join will consume all `Track` tuples before consuming any `Release` tuples.~~
- ~~The join will consume all `Release` tuples before consuming any `Track` tuples.~~

b. (4 marks)

Suppose instead that the DBMS executes this query using a hybrid hash join iterator with `Track` as the probe relation and `Release` as the build relation. As was the case for part (a), suppose that the access method for `Track` produces `Track` tuples in `Genre` order, and the access method for `Release` produces `Release` tuples in `ReleaseDate` order. Which of the following statements must be true? (Identify all that are true.)

- ~~The output of the join will be in `ReleaseDate` order.~~
- ~~The output of the join will be in `Genre` order.~~
- ~~The output of the join will be in `ReleaseID` order.~~
- ~~The join will consume all `Track` tuples before producing any output tuples.~~
- The join will consume all `Release` tuples before producing any output tuples. (**TRUE**)
- ~~The join will consume all `Track` tuples before consuming any `Release` tuples.~~
- The join will consume all `Release` tuples before consuming any `Track` tuples. (**TRUE**)

4 (9 total marks) Suppose that a database contains two relations, R and S. R has $50,000$ tuples and occupies 500 data blocks, and has a B+-tree index on attribute $R.a$. The index is unclustered and has a height of 2. The leaves of the index occupy 10 blocks. The relation S has $200,000$ tuples and occupies 2000 data blocks.

Suppose that the DBMS executes the following three queries in sequence, i.e., one at a time, in the order listed:

Q_1: `select * from R order by R.a`
Q_1 is executed by using the index to read the tuples of R in $R.a$ order.

Q_2: `select * from S`
Q_2 is executed by using a table scan of S.

Q_3: `select * from R order by R.a`
Q_3 is the same as Q_1, and is executed the same way.

a. (3 marks)
Assuming that the buffer cache is very small (no cache hits), how many disk block reads will the DBMS require to execute each of the three queries? Your answer should consist of three numbers, one for each query.

- Q_1: 11 index block reads plus 50000 data block reads (one per tuple) since the index is not clustered.
- Q_2: 2000 data block reads for the table scan.
- Q_3: 50011 - same as Q_1, since nothing has been cached.

b. (3 marks)
Repeat part (a), but this time assume that the buffer cache is large enough to hold 1200 blocks, and that the buffer replacement policy is LRU. Assume that the buffer cache is empty prior to the execution of Q_1. Again, be sure to give an answer for each query.

- Q_1: 11 index block reads plus 500 data block reads. Each data block will remain in the cache once it is read for the first time, and so will only be read once.
- Q_2: 2000 data block reads for the table scan.
- Q_3: 511 - same as Q_1. The cache will be full of blocks of S after the execution of Q_2, so Q_3 will need to re-read each block of R and R's index.

c. (3 marks)
Repeat part (a), but this time assume that the buffer cache is large enough to hold 1200 blocks, and that the buffer replacement policy is 2Q. (2Q uses a target size of 300 for its FIFO queue.) Assume that the buffer cache is empty prior to the execution of Q_1. Be sure to give an answer for each query.

- Q_1: 11 index block reads plus 500 data block reads. Each data block will remain in the cache once it is read for the first time, and so will only be read once.
- Q_2: 2000 data block reads for the table scan.
- Q_3: 0 Since Q_1 reads R's data blocks multiple times, those blocks will be promoted to the second queue by the 2Q algorithm. They will not be evicted from the cache when Q_2 runs, since the S blocks are read only one time and remain in the first queue.

5 (10 total marks) Suppose that a database contains a relation R stored in B blocks, with T tuples per block. The DBMS executes the following query:

```
select * from R where R.a > 10 order by R.a
```

Let f represent the fraction of R's tuples that satisfy the query predicate `R.a > 10`. For example, $f = 0$ means that none of R's tuples satisfy the predicate, $f = 0.5$ means that half of R's tuples satisfy the predicate, and $f = 1.0$ means that all of the tuples satisfy the predicate.

Suppose that the DBMS executes the query using the following plan:

1. use a table scan to read all of R, filtering out on the fly the non-matching tuples (those that do not satisfy the predicate).
2. sort the matching tuples, using an external merge sort, to produce them in `R.a` order.

a. (4 marks)

What is the I/O cost (number of disk block I/O operations) required to execute this plan? Indicate the cost of each iterator (table scan and sort) separately. Assume that the merge sort operator has sufficient memory to complete the sort using a single merging phase. Express your answer in terms of f and the other constants listed above.

The table scan will read B blocks of data, but after filtering will pipeline about fB blocks of data to the sort operator. During run formation, the sort operator will write fB blocks to temporary storage on disk. During the merge, the fB blocks will be read back in. Thus, the total I/O cost will be $B + 2fB$.

b. (3 marks)

Let M represent the amount of memory (measured in blocks) available to the merge sort operator. What is the minimum value of M required to ensure that the merge sort can complete the sort using a single merging pass? Express your answer in terms of f and the other constants listed above.

A total of fB blocks must be sorted. The length of each run will M, and so the number of runs will be $\frac{fB}{M}$. Merging these runs in one pass requires one block of memory per run being merged, so at most M runs can be merged in one pass. To ensure a single-pass merge, the sort operator requires

$$M \geq \frac{fb}{M}$$

or

$$M \geq \sqrt{fB}$$

c. (3 marks)

If R has an unclustered index on `R.a`, an alternative plan for executing the query would be to retrieve matching tuples in `R.a` order using the index, thus eliminating the sort operation. As f approaches 0.0, which of the two plans (table scan then sort, or index scan) do you expect will have the lowest cost? Briefly explain your answer.

Using the unclustered index will become the best plan as f approaches zero. As $f \to 0$, the cost of the original plan goes to B. As $f \to 0$, the cost of the unclustered index plan will approach 0.

6 (8 total marks) Provide brief answers to the each of the following questions. "Brief" means a sentence or two.

a. (2 marks)
Why does a DBMS buffer manager pin pages, and what effect does pinning have on the buffer's page replacement policy?

> Pages are pinned to indicate that they are currently in use by a worker. The buffer manager does not evict pinned pages.

b. (2 marks)
Many DBMS store data in files. Identify at least two ways that an underlying file system can impact the performance or correctness of the DBMS.

> - The file system controls the physical placement of data on the storage device(s).
> - The file system adds an additional layer of caching between the DBMS buffer cache and the storage device(s).
> - The file system may reorder I/O requests, and in particular write requests.

c. (2 marks)
Suppose that an iterator-based query plan includes hash join, and that the join has enough memory so that it only needs to partition its inputs once, i.e, there is no recursive partitioning. How many times will each tuple of the build input be written to and read from secondary storage by the hash join operator? How many times will each tuple of the probe input be written to and read from secondary storage by the hash join operator?

> Each tuple of the build relation will be written once to a temporary table on secondary storage and read back in once. The same is true for each tuple of the probe relation. Note: for the hybrid hash join variant, tuples in one partition of the probe relation and tuples in one partition of the build relation will never be written to (or read from) secondary storage. Tuples in the remaining partitions of both relations will be written and read once each.

d. (2 marks)
In pipelined query plans, some operators (like external merge sort) are called blocking operators. What does it mean for an operator to be a blocking operator?

> A blocking (unary) operator consumes all of its input tuples before producing any output tuples.

7 (3 marks) Suppose that a database occupies a total of P pages (blocks) on secondary storage. The database is stored on 10 disk drives, each of which cost \$100 and provides 100 IOPS (I/O Operations per Second). Suppose that, given the current price of memory and the size of the pages, an analysis like the one that led to the 5 Minute Rule determines that M ($0 < M < P$) of the database pages should be kept in memory, and the remaining $P - M$ pages should be kept on disk.

Now, suppose that the 10 disk drives that hold the database are replaced with a single SSD (flash memory based) storage device. This new device costs \$500 and provides 50,000 IOPS. (It is large enough to hold the entire database if necessary.) The 5 Minute Rule analysis is repeated for the new system, and the analysis determines that M_{SSD} ($0 < M_{SSD} < P$) of the database pages should be kept in memory. Assuming that the cost of memory and the database pages' access patterns have not changed, which one of the following must be true?

- $M_{SSD} = M$
- $M_{SSD} \geq M$
- $M_{SSD} \leq M$

Briefly justify your answer.

A Five Minute Rule analysis compares the cost of space in memory vs. the cost of bandwidth (IOPS) to secondary storage. In this example, the cost of memory stays the same, but the IOPS cost changes from \$1 per IOPS to \$0.01 per IOPS with the introduction of the SSD. Since the SSD is cheaper than disk, the SSD will be cost-effective for at least as many pages as the disk was. Thus, we should have at least as many pages on the SSD as we did on the disk, and therefore fewer pages in memory. The correct answer is $M_{SSD} \leq M$.