CS127 Quiz 2 Prep

Quiz 2: November 13th, 2019 3:00 P.M.

This ungraded handout is to help you prepare for Quiz 2. Answers will be posted later.

The warmup problems to HW4 and the solution to Problem 5 are also relevant to the quiz and should be reviewed with this quiz prep.

In answering these questions, if you don't understand the slides, the answers can be found in the textbook (except for the questions about columnstores and in-memory databases, which are only in the newer edition of the textbook. For those, review the slides, and if you still have questions, post on Piazza or come to office hours).

Topic #1 Storage and File Organization:

General goal: to gain a rudimentary understanding of storage and file organization so that you can understand how it affects database performance.

More specifically, understand

i block accesses and how they affect database performance

ii how fixed length tuples may be stored and maintained

iii how variable length tuples may be stored and maintained

iv how sequential file organization is laid out and maintained

v what multitable clustering is and when it is useful

vi what columnstores are and when they are useful

vii the advantages of in-memory databases

1. The sum of which two factors yields disk access time? Define each factor.

   Answer: Seek time and rotational latency. Seek time: the time it takes for an arm to move to the needed track. Rotational latency: the time it takes for the track to spin to the right sector.

2. What is a random access pattern, and why is it slower than a sequential access pattern?

   Answer: A pattern of access where each disk block is accessed from a random location relative to the last block. This increases access time, since there is likely a seek time and rotational latency between each block access

3. A given file stores n fixed-length records consecutively on disk blocks (if you are unfamiliar with the terms "file," "record," or "block" see the textbook, beginnings of section 10.2.3 and 10.5). A record at position i is deleted by the user. To maintain the file organization, we have a few options:

   a. Shift all records from position $i+1$ through $n-1$ to positions i through $n-2$

b. Move record at position $n-1$ to position i

c. Use a free list (see slides or textbook for definition of a free list).

Compare these three methods in terms of CPU resources, and explain why the last one is generally favored.

Answer: a. requires writing many records, and b. requires accessing the last record, which, in a sizable file will require another block access, while C requires only linking the free list to i's address. Since it requires the least operations, it is favored.

4. You insert a fixed length tuple into a relation that uses a free list. Describe what happens when a tuple is inserted

   Answer: The DBMS follows the free list header to the address of the first empty record, and inserts new tuple there, and updates the header to point to the next empty record formerly pointed to by the first empty record.

5. What are the three types of data held in a slotted page structure header?

   - The number of entries
   - the end of the free space
   - an array whose entries contain the location and size of each record

6. When it comes to slotted page structures, external pointers point to the entry in the header that contains the location of the record, rather than to record entries directly, to allow records to be reorganized without updating their respective external pointers. Why is this organizational flexibility important in the slotted page structure?

   Answer: Since slotted page structures are used for variable length records, it's important to shift around the records to fill in empty space whe a record is deleted since otherwise the structure becomes fragmented and wastes space.

7. a. If files are organized by sequential storage, what happens when you attempt to insert a record in a location that does not have any free space?

   Answer: It gets put into an overflow block, and the pointers in each record are adjusted accordingly.

   b. Given the above, what must the system do periodically to maintain sequential order of files?

   It must do a clean up operation to reorganize the files to be in sequential order

8. When is the buffer manager involved in "pinning" a block?

   Answer: When the block is not allowed to be written back to disk, for example, when it is under a write lock, so as to prevent inconsistencies.

9. Given the following relation:

addr	dept_name	emp_id	sal
0	accounting	0	100000
1	actuarial	2	120000
2	engineering	3	125000
3	engineering	1	132000

   draw how the relation is represented in columnstore (i.e. column-oriented storage)

   Three columns: one of all the values of dept_name, one of all the values of emp_id, and one of all the values of salary

10. How does columnstore make file compression easier?

    Since each column is stored contiguously as a separate unit, we can use compression technique that is most suitable for the particular column depending on the column data type and whether the column is sorted or not.

11. Updates or deletions of single tuples from column stores are especially expensive. What feature of column stores makes deletes and updates of single tuples especially expensive and why?

    Answer: The compression of columns makes updating or deleting a tuple require rewriting all the subsequent compressed values in its compression group, which is expensive. Since data analysis does not generally involve single attribute updates or deletions (rather, they use batch inserts and deletes), this is not a problem for data analysis databases.

12. (a) Is a buffer manager necessary for in-memory databases? Explain.

    No it is not, since the buffer manager manages which blocks are read in from disk to main memory and which are kicked out of memory to make room for other reads, and this is not necessary if the entire database is in memory.

    (b) What flexibility is gained from having an in-memory database?

    No more concern about sequential access vs random access of blocks, since all is in memory.

Topic #2 Indexes

General goals: To understand

i. clustered vs non-clustered indexes

ii. dense vs sparse indexes

iii. the advantages and use applications of each

1. What are the advantages of sparse indices over dense indices? The advantages of dense indices over sparse indices?

   Sparse:

   - Takes less storage space, so can be loaded into memory more easily.
   - When a tuple is inserted, updated, or deleted, may not need to change a sparse index.

   Dense:

   - Direct lookup access.

2. What condition must be met for a sparse index on a relation to be feasible and why?

   The sparse index must be a primary/clustered index, in other words the relation must be sorted by the search key of the sparse index, because the sparse index only indexes directly into intermittent entries, and depends on scanning forward from that entry to find the search key.

3. You have a dense index that is too large to fit in memory and that has few repeat values. You decide to write an outer index on it. What type of outer index should you write: dense or sparse, and why?

   You should write a sparse outer index, because a dense one will not cut down the number of entries significantly, since there are few duplicate values in the inner index, and your whole point in writing the outer index is to make it small enough to fit into memory. You may use a sparse index, since the entries of the inner index are already sorted.

4. Most of your queries of the relation employee(emp_id, emp_name, dept_id, mgr_id, salary) are of the following form:

```
Select * from employee where emp\_id = Y
Select * from employee where dept\_id = X
```

Taking search time as your sole optmizing goal, and considering the potential cardinality of each search, which search key should you designate for your primary index?

Considering that emp_id is a primary key, the first query above returns just a single tuple, whereas the second query may return a list of tuples, since many empoyees may work in a single department. You will therefore benefit from having a primary index on dept_id, since that will allow all tuples of the same value to be returned in a sequential access pattern. It doesn't benefit you to make emp_id you primary index, since your queries on emp_id only return single tuples anyway.

Topic #3 B+-Trees

General goals: To understand

i. the B+-Tree algorithm

ii. how B+-Tree indexes enhance index performance

iii. how they compare to sequential indexes

According to the textbook, there are three rules of B+-Trees of max degree n:

1. The root node may have zero, or between 2 and n (inclusive) children
2. Non-root internal nodes may have no less than $\lceil n/2 \rceil$ children
3. Leaf nodes may have no less than $\lceil (n-1)/2 \rceil$ keys (Note: the visualization of B+-Trees online does not adhere to this last rule. We do not know the source of its algorithm, so cannot vouch for its authenticity. Please see Stan's slides and the textbook for an authoritative algorithm.)

   (*Note 2: The stencil code for the B+Tree project may vary slightly from the above constraints (like floor instead of ceiling). However, for this quiz prep, please use the above constraints.)

Every node

1. Given the above constraints, construct a B+-Tree index where $n = 4$ from the following key values: 4, 5, 6, 3, 11, 15, 12, 2, 0, 7, 1, 9, 10

   Your tree construction should be in accordance with the B+-Tree insertion algorithm given in the slides and textbook. (Note that the textbook and slide's convention is that if you add a fourth key to a full node, requiring a split, you push up the third key, rather than the second). Assume that the tree is initially empty and that the values are added in the above order. Draw the result. Show all pointers in the result (including pointers to records). There is only one correct solution.

   Answer: See figure 1

2. Now delete the following keys from your tree, in the following order: 15, 12, 9. Draw the result.

   Answer: See figure 2

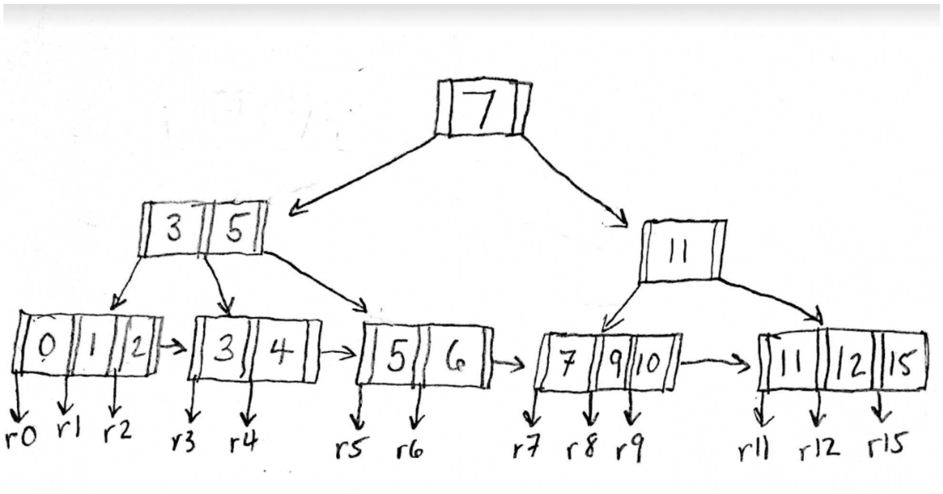

Figure 1: After adding all keys. Empty entries in each node have been omitted for clarity's sake

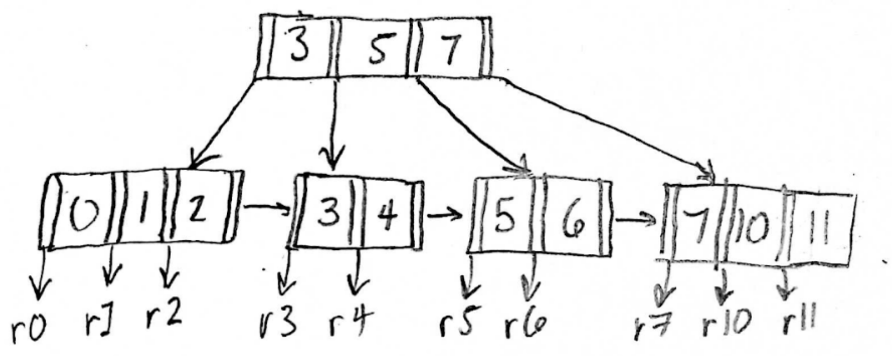

Figure 2: After deleting 15, 12, 9. Empty entries in each node have been omitted for clarity's sake

3. If you did the above deletion correctly, when deleting 9, you merged the node containing 7 with the node to its right, containing 10 and 11, rather than with the node to its left, containing 5 and 6. Why was the node to its right the correct node to merge with, rather than the node to its left?

   Answer: Because the algorithm says that if a node has underflow so much be merged, then merge with the other children of its parent, and the node containing 10 and 11 is a child of the parent node of 7, while the node containing 5 and 6 is not.

4. The number of nodes traversed to search a B+-Tree is at worst

   $$\lceil log_{\lceil n/2 \rceil}(N) \rceil$$

   where n is the number of pointers (the branch out factor) and N is the number of records in the database.

   (a) Explain why this is the worst case scenario and what the equation for the best case scenario is

   This is the worst case scenario, because it reflects a B+-Tree that has only $\lceil n/2 \rceil$ pointers per internal node, which is the least possible pointers allowed. Best case scenario would have

   $$\lceil log_{\lceil n \rceil}(N) \rceil$$

   nodes to traverse, which is when every node has maximum children.

(b) Typically each node is made to be the same size as a block, which is typically 4096 bytes. If pointers are 8 bytes (for example, 32 bit integers), and the search key is 32 bytes (for example, a value of domain CHAR(32)), what is the maximum possible value of n?

4096 / (32+8) = 102.3. So we can store 102 key-pointer pairs for pointers 0 through 101, plus 12 extra bytes for pointer 102. This gives us $n = 102$

(c) Suppose that you set n to the maximum size you found above. If so, what is the worst case number of blocks that you must access in your B+-Tree to find a given key in if there are 1 million records in your database?

$\lceil log_{51}(10)^6 \rceil = \lceil 6 * log_{51}(10) \rceil = \lceil 3.52 \rceil = 4$

(d) What is the worst case number of blocks that you must traverse if the database grows by a factor of 1000 to 1 billion records?

Using the same math, we get worst case is 5 blocks.

(e) What does this tell you about the scalability of read-databases when using B+-Trees?

Your database can grow exponentially, and the B+-Tree search time will hardly increase, making B+-Trees very good for scalable databases.

5. The hierarchical structure of B+-Tres and multilevel sequential indices looks similar, but has this key difference: B+-Trees are broken into nodes, with pointers between the nodes (as in a linked list), whereas each level in a multilevel sequential index is laid out sequentially in memory (with overflow blocks when needed). How does this difference account for B+-Trees being better suited for dynamic databases?

   This difference means that when you insert keys in a B+-Tree, even if there is not room in the node for the key, you can just add a node, and don't have to shift the whole index over, since sequential layout is not required between nodes.

6. How can sequential indexing methods compensate for this disadvantage?

   They can be initialized with a set amount of free space between groups of records, for example 20%, so that when records are added or deleted, the whole index does not have to be shifted. But this only works so long as the free space is not used up.

7. How do B-Trees differ from B+-Trees?

Topic #4 Hashing

General goals: To understand

i. what makes a good hash function

ii. what considerations must go into making a static hashing index vs a dynamic hashing index

iii. dynamic hashing algorithms

iv. when it is beneficial to use hashing as opposed to other (previously discussed) indexes

1. (a) A good hash function distributes values uniformly from the set of all possible values. Why is this important?

   So that values will not accumulate unevenly and cause unnecessary bucket overflow.

   (b) A good hash function distribute keys randomly, that is, such that the result of the hash distribution appears random and does not preserve patterns in the records inserted. Why is this important?

   So that if there is a pattern in input data that could skew the data into a narrower range of buckets (for example, many more keys in a certain range of values than in other ranges of values) that pattern will be disrupted so records can be distributed evenly across buckets.

2. (a) If you want to build a static hash index on your database, what should you know to prevent the hash index from soon becoming slow?

   You should have a prediction of how large your database will be, so that you can allocate enough buckets to it so that there will be minimal overflow.

   (b) Is that information necessary if you want to build a dynamic hashing index? Why not?

   No, since dynamic hashing algorithms increase and decrease the number of buckets dynamically as the size of the database changes.

3. Review your solution to HW4 problem 5 and the correct answer (after it is released). Be sure you know how to do extendible and linear hashing.

4. When would it make sense to create a hashing index for your database, instead of a B+-Tree index or sequential index?

   When you know that the range queries are unlikely.