1 Storage [24 pts]

1. What is “non-volatile” storage? [3 pts]

**Answer:** Non-volatile storage retains its contents even if it is not powered.

Circle the ones that are non-volatile.

- Hard Disk
- Main Memory
- Cache
- Magnetic Tapes

**Answer:** Hard disk and magnetic tapes are non-volatile.

2. For a disk, the data transfer rate is 500MB/s, the average seek time is 5ms, and the average rotational latency is 5ms. How much time would it take to write 100 blocks of data, each of size 10KB, randomly vs sequentially? For sequential case, don’t forget the initial seek. Underline the final answer. [3 pts]

**Answer:** Sequential Case: initial seek = 5ms. We then have to wait for 5ms (rotational latency) for the appropriate sector to come under the head. We then write 1MB of data, which would take 1000/500 = 2 ms. So a total of 12ms.

Random case: We need 100 * (5 + 5) = 1000ms for the seeks and the waits (because of rotational latencies). The writing part itself is still going to be 2ms. So a total of 1002ms.

3. In disks, typically the amount of data that can be stored on the outer tracks is higher than the amount of data stored in the inner tracks (since the outer track is longer, there is more space). More specifically, the amount of data per track is proportional to the circumference of the track. Consider a disk with \( n \) tracks, with the \( i \)th track at a distance \( i \times c \) from the center, where \( c \) is a constant \( (i = 1, \ldots, n) \). Say the 1st track (closest to the center) holds 1GB of data. Derive an expression for the total capacity of the disk (assume a single platter). [3 pts]

**Answer:** \( \frac{n(n+1)}{2} GB \)

4. What is the purpose and main advantages of “non-volatile write buffers” in disks? [3 pts]

**Answer:** They are used to buffer the writes to disks, so that the operation succeeds immediately.

5. List three of the tasks of a disk controller. [3 pts]

**Answer:** (1) remapping bad sectors, (2) buffering writes to reduce the latency, (3) checksums to verify correctness of blocks.
6. Consider a RAID 1 set up with 8 disks, and a RAID 5 setup with 5 disks. Say the data transfer rate is 100 MB/s. Consider a workload of the following type: the application is going to read a large number of blocks (say 1,000,000) and will modify 10% of the blocks (so 100,000 blocks will be modified). Assume all read/writes will be sequential, so the data transfer rates of 100MB/s can be sustained for all disks. Compare the steady state performance of the two RAID setups, in terms of how much time it takes each setup to process (read, modify if needed, write) say 100,000 blocks. For simplicity, assume blocks are of size 10KB each. [4 pts]

Answer: 100 MB/s = 10,000 blocks/s. With RAID 1, the aggregate transfer rate is 80,000 blocks/s, and with RAID 5, it is 50,000 blocks/s.

Over a large period of time, say we process 100,000 blocks of data, requiring 100,000 reads, and hence 10,000 writes. In RAID 1, the reads can go to either copy, but the writes must go to both. So total number of block I/Os = 120,000, which will take 120,000/80,000 = 1.5 seconds.

With RAID 5, each write will involve reading the parity block, and modifying both the original block and the parity block. So total number of block I/Os = 100,000 + 40,000 = 140,000, which will take 140,000/50,000 = 2.8 seconds.

7. Briefly describe the slotted page structure for storing records in pages, and discuss its main advantages over the alternatives. [5 pts]

Answer: See notes or the book.

2 B+-Tree/Indexes [22 pts]

Figure below shows a secondary B+-Tree index over a relation with search key containing two attribute (A, B). So the keys are first sorted by A, and then by B (if the value of A is the same). Assume A and B are integer attributes. The order of the B+-Tree is 3 – each node holds at most 3 key values (and 4 pointers) and at least 2 key values (except root). The number of tuples that fit in each disk page is 3 (as shown).

The actual relation pages are not shown. Use e.g. RPage-1(8, 3), RPage-2(8, 3) to refer to the relation pages containing the tuples corresponding to the first and second occurrences of the search key (8, 3) in the index respectively (recall that this is a secondary index).

Answer the following questions for this tree.

1. Enumerate the pages accessed and actions taken when answering the query \( \sigma_{(A=9) \land (B=20)}(R) \). [2 pts]

Answer: 1. BPage 1: Follow the third pointer to BPage 4.
2. BPage 4: Follow the first pointer to RPage-1(9, 20).
3. \textit{RPage-I}(9, 20): Return the tuple \((9, 20, \ldots)\).

2. Enumerate the pages accessed and actions taken when answering the query \(\sigma_{(A=9) \land (B<30)}(R)\). [3 pts]

\textbf{Answer:} 1. \textit{BPage 1}: Follow the second pointer to \textit{BPage 3}.
2. \textit{BPage 3}: Follow the pointer to \textit{RPage-I}(9, 10); then follow the pointer to \textit{BPage 4}.
3. \textit{RPage-I}(9, 10): Return the tuple \((9, 10, \ldots)\)
4. \textit{BPage 4}: Follow pointer to \textit{RPage-I}(9, 20); after coming back, terminate search.
5. \textit{RPage-I}(9, 20): Return tuple \((9, 20, \ldots)\).

3. Enumerate the pages visited and actions taken when inserting a new tuple \((8, 2, \ldots)\). Draw the new index structure. Assume the tuple has already been inserted into the relation page, and we have a pointer to that relation page, \textit{rpage}. [5 pts]

\textbf{Answer:} [Diagram of index structure]

4. Draw the new index structure after deleting the tuple corresponding to the key \((9, 20)\) from the original index. [3 pts]

\textbf{Answer:} [Diagram of updated index structure]

5. Let the height of a real instantiation of the above index be 4. Approximately estimate the cost of executing the query \(\sigma_{(A \in [10, 15, 31, 43]) \land (10 < B < 50)}(R)\) in seconds; let \(t_s\) denote the cost of a seek, \(t_b\) denote the cost of a block transfer. Let \(n\) denote the number of matching records, and let \(d\) denote the order of the B+-tree (average number of pointers per node). State any assumptions made. [5 pts]

\textbf{Answer:} (a) Since this is a secondary index, we will have \(n\) random I/Os for fetching the actual records.
(b) For \( i = 10, 15, 31, 43 \), we must seek to the page containing \( (i, 10) \). We will need to do 4 random I/Os for each of those, so that gives us at most 16 random I/Os (4 for each).

(c) Finally, we need to access an additional about \( n/d \) index pages (for each \( i \), we need to scan forward from the page containing \( (i, 10) \) to the page containing \( (i, 50) \)). Although it is possible to make those sequential, let's assume they are also random I/Os (in the naive implementation, we will go the RPages in between).

(d) The total cost then is: \( (n + 16 + n/d) \times (t_s + t_b) \).

6. (Not related to the above index) Consider a relation \( R \) that is 3.2GB in size. Also, let the block (disk page) size be 4KBytes. Assume the page utilization (how full the pages are) is 80% (for all pages including index pages). Compute the height of a B+-Tree index on this relation, assuming that the maximum number of \((key, pointer)\) pairs that can be stored per page is 125. Clearly show your calculations. [4 pts]

**Answer:** 3.2GB occupies \(3,200,000,000/4,000\) blocks = 800,000 blocks.

The average number of pointers per page = \(125 \times 80\% = 100\).

The number of pages at the leaf level = \(800,000/100 = 8,000\).

The number of pages at the next level = \(8,000/100 = 80\).

We need one more page at the root level = 1.

So the height of the index is 3.

3 Query Processing [24 pts]

1. Briefly explain the main difference between Hybrid Hash Join operator and the normal Hash Join operator, and why the former is better than the normal Hash Join operator. [3 pts]

**Answer:** Hybrid hash join keeps the first partition in memory while partitioning the smaller relation. That way it saves the cost of writing that partition to disk, and reading it back in.

2. List three of the tasks of the query parser. [3 pts]

**Answer:** (1) Resolve references, (2) Syntax check, (3) Create a relational algebra expression from the query, (4) Infer new predicates, . . .

3. Main Memory = 1000 blocks. We are asked to sort a relation of size 3000 blocks, and write the sorted relation back to the disk. Enumerate the steps in sorting this relation using external merge-sort, and carefully count the number of seeks required. [5 pts]

**Answer:**

(a) Read the relation and create 3 sorted runs. For creating each run, we need a seek to the input relation to read the blocks, and a seek to the output relation to write the run. Total 6 seeks.

(b) In the merge phase: if we allocate 4 buffers, one for each sorted run, and one for the output: then we pretty much need a seek to read each of the blocks of the runs, and each block of output. So total number of seeks is: 6000.
4. Design techniques for reducing the number of seeks required for the merge-sort. If your answer above already minimizes the number of seeks, explain. Hint: Judiciously control how the writes are done and how many buffers are allocated to each relation. [5 pts]

**Answer:** We modify the Step 2 as follows: At the beginning, read 250 blocks from each of the sorted runs, and assign the remaining 250 blocks to the output. In other words, we do I/Os in units of 250 blocks each. The number of seeks would drop to $4 \times 3 + 12 = 24$ (+ 6 for creating the sorted runs).

This is actually not optimal. The total number of seeks can be reduced further to maybe about 12 by reusing the empty blocks from the runs to hold the output buffers.

5. Recall that given two relations $R(A, B)$ and $S(B)$, the division operation, $R \div S$, produces a relation $T$ with schema $T(A)$ such that $T \bowtie S \subseteq R$. For example, if $S = \{(b_1), (b_2)\}$, then $T$ (the result) will contain $(a_i)$ if and only if both $(a_i, b_1)$ and $(a_i, b_2)$ exist in $R$.

Design an algorithm to implement this operator, and very briefly analyze its cost. You can assume that $S$ fits in memory. [5 pts]

**Answer:**

(a) Sort $S$ on $B$, and keep it in memory. Sort $R$ on the key $(A, B)$ (i.e., sort it on $A$ first and then on $B$). For both, eliminate duplicates. Scan $R$ in order. For every distinct value of $A$, say $a_i$, compare to see if there is a tuple $(a_i, b_j)$ for every value $b_j$ in $S$. Since both relations are sorted, this can be done quite efficiently by simply going down both lists simultaneously.

(b) A hash-based solution can be found in the solution to the practice exercise 13.8 (at http://db-book.com).

6. Can you do the division operation more efficiently, without sorting $R$, if you knew that $S$ contained a relatively small number of tuples, say about 256 tuples? [3 pts]

**Answer:** Build another small hash table on $S$, using which map each of the values of $S$ to a number between 0 to 255.

Instantiate a hash table on $R$. With each entry there, keep a bitmap of length 256. Initialize the bitmap to 0.

For each tuple $r \in R$, find the bit position for $r.b$ using the hash table on $S$. If the tuple already exists in $R$, update the bitmap to set that bit to be true. If the tuple doesn’t exist, add it to the hash table.

At the end, go over all tuples in the hash table on $R$, and check if the bitmap is all 1’s.

This will be more efficient than the sorting-based option if the number of tuples in the relation $R$ is small enough that the hash table fits in memory, but is not so small that the entire $R$ fits in memory.