W4111_003_2024_3 – Introduction to Databases Homework 3A

Nathan Gong

Overview

HW3A is primarily questions with short, written answers. Please be concise. Please see Ed for the details on submission date and submission format.

Storage Devices

S1. What are three important criteria for choosing the type of storage to use for data?

1. Speed with which data can be accessed from storage
2. Cost per unit of data in storage
3. Reliability of the storage

S2. Some databases use magnetic disks in a way that only sectors in outer tracks are used while sectors in inner tracks are left unused. What might be the benefits of doing so?

A benefit of using only sectors in outer tracks of magnetic disks is reduced seek time, as the disk arm is optimized for quicker access to outer tracks. Another benefit is minimized rotational latency, since magnetic disks may suffer from increased latency when accessing inner tracks due to the higher track density leading to more time taken for the accessed sector to appear under the head. Additionally, since the outer tracks have a larger circumference, each disk rotation covers more data, correlating to data transfer rates ranging from 25-200 mb/sec, which are higher than those of inner tracks.

S3. Assume that the access pattern to a relation is primarily sequential. How might the storage manager place the relation's data on a cylinder, head, sector storage device?

If the access pattern to a relation is primary sequential, the storage manager would place the relation's data such that successive requests are for successive disk blocks, and disk seek is required only for the first block. This might entail storing the data in sequential tracks within the same cylinder, within a single head or small number of heads, and on sequential sectors on a

single platter. As such, once the disk head is positioned at the start of the relation's data, it would be able to read sequentially, minimizing seek time and rotational latency.

S4. List two advantages of solid-state storage/drives over magnetic disk drives/storage devices. What is a disadvantage of SSD relative to magnetic disk storage?

Advantages:

1. Solid-state drives have faster data access and transfer speeds than magnetic disk drives due to the use of flash storage
2. Solid-state drives have better durability than magnetic disk drives since they do not contain moving parts such as spinning disks and heads

Disadvantages:

1. Solid-state drives have a limited number of write and erases per block before becoming unreliable which causes them to have lower endurance than magnetic disk drives.

S5. What is an advantage of RAID-0 relative to RAID-5? What is an advantage of RAID-5 relative to RAID-0?

An advantage of RAID-0 relative to RAID-5 is that RAID-0 has faster performance than RAID-5 and can be used in applications where data loss is not critical. An advantage of RAID-5 relative to RAID-0 is that RAID-5 has better data redundancy than RAID-0 due to partitioning data and parity among all disks.

S6. Briefly explain logical block addressing and cylinder-head-sector block addressing.

Logical block addressing is used by solid-state drives and assigns a linear address to each block on the disk. Cylinder-head-sector block addressing is used by magnetic disk drives and addresses data based on the specified cylinder, read/write head, and sector.

S7. Briefly explain the disk block access concepts of read-ahead and disk-arm scheduling.

Read-ahead scheduling involves reading extra blocks from a track in anticipating that they will be requested soon. Disk-arm scheduling involves reordering block requests so that disk arm movement is minimized.

S8. Briefly explain the concepts of network attached storage and storage area networks.

In network attached storage, a file system interface using networked file system protocol is provided instead of a disk system interface. In storage area networks, a large number of disks are connected by a high-speed network to a number of servers.

Storage Formats

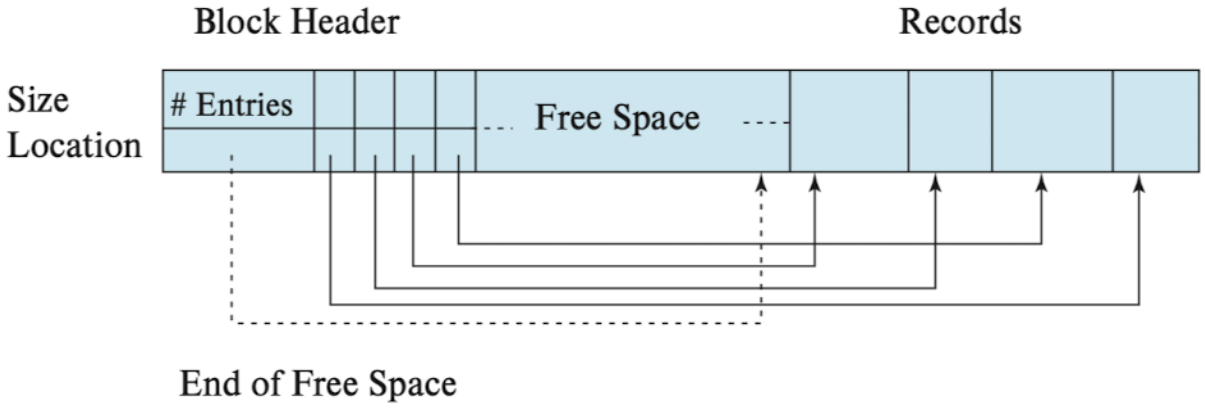
F1. Assume the storage manager is using fixed size/length records. Explain two approaches to implementing addition/deletion of records.

One approach to implementing addition of fixed size records to store record i with size n starting from byte $n(i-1)$. Another approach to implementing addition of fixed size records is to not allow records to cross block boundaries.
One approach to implementing deletion of records is to move records $i+1, \ldots, n$ to $i, \ldots, n-1$. Another approach to implementing deletion of records is to not move records but link all free records on a free list.

F2. VARCHAR is a common column type and is by definition not a fixed size field. Why might the storage manager choose to use fixed size records and allocate the maximal record size for records with VARCHAR fields?

The storage manager may choose to use fixed size records for VARCHAR fields for simplified memory management. The storage manager can more easily manage offsets within records and make data retrieval faster. The logic for inserting, updating, and deleting records is also simplified since the storage manager does not need to check how much space the VARCHAR field will occupy.

F3. Consider the image below. Why does the storage manager consolidate free space by moving records when a deletion occurs?



The storage manager moves records when a deletion occurs to keep the records contiguous with no empty space between them. The block header entry must also be updated to keep track of the end of free space in the block and the location and size of each record. This results in improved read/write performance and more efficient use of storage space.

F4. Briefly explain *heap file organization, sequential file organization and multi-table clustering organization.* Give a scenario or use case for choosing each of the approaches.

In heap file organization, records can be placed anywhere in the file where there is free space and usually do not move once allocated. Heap file organization can be implemented using a free-space map. A use case of heap file organization is when record storage needs to be quick and simple and there is no need for sorting or other complex queries.
In sequential file organization, records are ordered by a search-key. A use case of heap file organization is for applications that require sequential processing of the entire file.
In multi-table clustering organization, several related tables are physically stored together in a single file based on some common attribute. A use case of multi-table clustering organization when queries are often used that involve multiple joins on relations that share some common attributes.

F5. Consider a relation/file that records credit card transactions. Applications periodically retrieve records. 90% of the retrievals are for transactions that occurred in the prior six months. What technique would you recommend for storing the records on disks/files?

I would recommend using sequential file organization for storing records of credit card transactions on disks/files. Since the majority of retrievals are for transactions that occurred in the prior six months, storing the records in a sequential order allows for quick and efficient access to the most frequently queried records. Table partitioning can also be used to separate records into smaller relations based on biannual time periods. This would allow applications that periodically retrieve records to have quicker access to the desired records in the past six months. This would also result in reduced costs of free space management and more efficient storage of older data on different storage devices.

Buffer Management

B1. Operating systems almost exclusively use the least recently used algorithm for the replacement policy. Why do databases sometimes use other algorithms? What is an example of a query for which most recently used might be a good replacement algorithm?

Databases sometimes use other algorithms besides the least recently used algorithm for the replacement policy when the assumption does not hold that data accessed recently is more likely to be used again soon. An example of a query for which most recently used might be a good replacement algorithm is a query that accesses sequential data, such as selecting records from a table where the date of the record falls in between a specified range.

B2. Briefly explain the Clock Algorithm for buffer replacement. What would motivate using the Clock Algorithm instead of least recently used?

In the clock algorithm, a pointer cycles through buffer blocks in a circular manner. Each buffer block has a reference bit that is set to 1 when the page is accessed and periodically reset to 0 when the page has not been accessed for a while. When a page needs to be replaced, the pointer checks the reference bit of the buffer block. If it is 1, the reference bit is cleared. If it is 0, the page is replaced. The clock algorithm would be used instead of least recently used for increased efficiency since it only requires one reference bit per buffer block, as well as simplified implementation using a circular queue structure. This can scale better for large and real-time systems.

B3. What is a pinned block? Does the buffer manager typically use pinned blocks when selecting a block for replacement? Why?

A pinned block is a memory block that is not allowed to be written back to disk. The pin is done before reading/writing data from a block, and the unpin is done when the read/write is complete. The buffer manager does not typically use pinned blocks when selecting a block for replacement since pinned blocks cannot be replaced until they are unpinned. Pinned blocks may contain data that is being modified, and replacing a block that is being actively modified could lead to loss of data integrity and consistency in the system.

B4. Why would a buffer manager not necessarily write updated blocks to disk in the same order that the updates occurred in the buffer?

A buffer manager might not write updated blocks to disk in the same order that the updates occurred in the buffer for better data consistency and writing efficiency. For example, buffer managers may write blocks to a nonvolatile RAM or log disk immediately for the purpose of data recovery in case of a crash. Additionally, since writing data to disk is relatively slower than accessing it, the buffer manager may accumulate several block updaters in the buffer and before writing them to the disk altogether. The buffer manager may also replace blocks according to its replacement policy, and the order in which blocks are written to disk may not necessarily match the order that the updates occurred if the updated blocks are replaced before being written to disk.

Indexes

I1. How many clustered indexes can exist on a relation/table? Is it possible for a sparse index to be non-clustering/unclustered?

A relation can only have one clustered index since it is the index whose search key specifies the sequential order of the file, and a file can only have one sequential order. It is possible for a sparse index to be non-clustering. For example, non-clustering sparse index could include only non-null values and point to the corresponding rows in a table while not affecting the sequential order of the rows in the table. Additionally, a multilevel index involving a sparse index on top of a dense index may be used.

I2. Assume that a very common query on a table with information about people is *select * from people where last_name like "XYZ%"* where X, Y and Z are characters. For example, queries of the form *where last_name like "FER%" or last_name like "HAW%"* Can you use a hash index to optimize performance for this type of query?

No, a hash index cannot be used to optimize performance for a query involving string pattern matching. A hash index is based on hashing the values of the indexed columns into unique buckets. This is efficient for exact match queries but is not suitable for range queries that would require scanning through rows in a sequential manner due to hashing not being ordered in a way that supports pattern matching.

I3. If indexes are so effective at improving query performance, what is the disadvantage of creating very many indexes on a table?

The disadvantage of creating many indexes on a table is that this results in poor performance on write operations. When rows are inserted/updated/deleted, all indexes on the corresponding table must be updated. If many indexes are created for that table, write operations become increasingly inefficient. Another disadvantage of creating many indexes on a table is that this can consume large amounts of storage, as each index consumes storage space. With large tables where each index can require a significant amount of storage space, the storage requirements for a table become increasingly large if it has many indexes created.

I4. Data Structure courses teach binary search trees. It is common for the degree of a B+ tree to be more than 2. Why do B+ tree use degrees higher than 2? What determines the degree?

B+ trees use degrees higher than 2 to improve index performance by reducing the height of the tree compared to an ordinary binary search tree. This can greatly increase the efficiency of write operations by restructuring the index in logarithmic time.
The degree of a B+ tree is determined primarily by the block size. A typical B+ tree node contains a search-key value and a pointer to a child/record. Thus, the degree is approximately the block size divided by the sum of the key size and the pointer size. Furthermore, each node that is not a root or leaf has between $\frac{n}{2}$ and n children, while a leaf node has between $\frac{n-1}{2}$ and $n-1$ values. If the root is not a leaf, it has at least 2 children. If the root is a leaf, it can have between 0 and $n-1$ values.

Query Processing

Q1. Briefly explain this sentence, "Each relational algebra operation can be evaluated using one of several different algorithms." Give three examples for a select operation.

This sentence means that for any relational algebra operation, there are several ways to implement that operation depending on the underlying database system, the types of indices available, and the query optimization strategies used. The implementation that is ultimately chosen for any operation tends to be the one that results in the lowest query cost. Three examples of algorithms that can be used to evaluate a select operation are a file-based scan using a linear search, an index-based scan using a clustering index based on key equality, and a comparison-based scan using a clustering index based on comparison of a selected column value that the relation is sorted on.

Q2. Consider the following strange query – *select * from customers join employees on customers.last_name != employees.last_name.* Would an index nested loop join work for this type of join?

No. While it could technically be possible to use an index nested loop join for this type of non-equality join, it would not work well. Index nested loop joins iterate through each row in the outer table and use an index on the inner table to find matching rows based on an equality condition, which works efficiently for joins based on equality conditions because indexes are used to find equality matches. However, non-equality conditions require iterating through the entire inner table to select rows that match the non-equality condition, which can lead to inefficient performance.

Q3. What is the primary advantage of a *merge join?* If a merge join requires sorting the relations before performing the join, how is it possible better than a nested-loop join.

The primary advantage of a merge join is that each block only needs to be read once, assuming that all tuples for any given value of the join attributes fit in memory. This results in linear time complexity for a scan through sorted data.
While a merge join requires sorting the relations before performing the join, it can sometimes be better than a nested-loop join in certain scenarios. Nested-loop joins perform a brute-force comparison of every tuple in both relations with a $O(mn)$ time complexity, while merge joins involve sorting in $O(nlogn)$ time and scanning both sorted relations in $O(m+n)$ time, which can be faster than nested-loop joins. Furthermore, if the relations are already sorted or indexed on the join key, merge joins become even more efficient than nested-loop joins.

Q4. Consider the query *select first_name, last_name from person residence_country='USA'.* What modification to the query might cause the query engine to create an index on *(first_name, last_name)?*

The query engine would create an index on (first_name, last_name) if these columns were used as a search condition such as a WHERE clause or a JOIN clause, or if operations were performed on these columns such as GROUP BY or ORDER BY. For example, the query engine may create an index on (first_name, last_name) for the following query:
select first_name, last_name from person where first_name='Nathan' and last_name='Gong' and residence_country='USA';

Q5. Briefly explain the concepts of pipelined evaluation and materialized evaluation.

In pipelined evaluation, several operations are evaluated simultaneously and the results of one operation are passed onto the next. Pipelines can be executed in demand driven evaluation where the system repeatedly requests the next tuple from the top-level operation, and producer driven evaluation where operators produce tuples eagerly and pass them up to their parents. For pipelining to be effective, evaluation algorithms are used that generated output tuples as tuples are received for inputs to the operation. Pipelining may not always be possible but is much cheaper than materialized evaluation since there is no need to store a temporary relation to disk.
In materialized evaluation, each operation is evaluated one at a time, starting at the lowest level. Intermediate results materialized into temporary relations are used to evaluate the next level of operations. Materialized evaluation can make use of double buffering, where two output buffers are used for each operation and one can be written to disk when it is full while the other is getting filled, resulting in reduced execution time. Although the cost of writing intermediate materialized results to disk and reading them back can be high, materialized evaluation is always applicable, unlike pipelined evaluation.