9

STORING DATA: DISKS AND FILES

Exercise 9.1 What is the most important difference between a disk and a tape?

Answer 9.1 *Tapes* are sequential devices that do not support direct access to a desired page. We must essentially step through all pages in order. *Disks* support direct access to a desired page.

Exercise 9.2 Explain the terms *seek time, rotational delay*, and *transfer time.*

Answer 9.2 Answer omitted.

Exercise 9.3 Both disks and main memory support direct access to any desired location (page). On average, main memory accesses are faster, of course. What is the other important difference between the two (from the perspective of the time required to access a desired page)?

Answer 9.3 The time to access a disk page is not constant. It depends on the location of the data. Accessing to some data might be much faster than to others. It is different for memory. The time to access memory is uniform for most computer systems.

Exercise 9.4 If you have a large file that is frequently scanned sequentially, explain how you would store the pages in the file on a disk.

Answer 9.4 Answer omitted.

Exercise 9.5 Consider a disk with a sector size of 512 bytes, 2000 tracks per surface, 50 sectors per track, five double-sided platters, and average seek time of 10 msec.

1. What is the capacity of a track in bytes? What is the capacity of each surface? What is the capacity of the disk?

2. How many cylinders does the disk have?
3. Give examples of valid block sizes. Is 256 bytes a valid block size? 2048? 51,200?
4. If the disk platters rotate at 5400 rpm (revolutions per minute), what is the maximum rotational delay?
5. If one track of data can be transferred per revolution, what is the transfer rate?

Answer 9.5 1.

$$bytes/track = bytes/sector \times sectors/track = 512 \times 50 = 25K$$

$$bytes/surface = bytes/track \times tracks/surface = 25K \times 2000 = 50,000K$$

$$bytes/disk = bytes/surface \times surfaces/disk = 50,000K \times 5 \times 2 = 500,000K$$

2. The number of cylinders is the same as the number of tracks on each platter, which is 2000.
3. The block size should be a multiple of the sector size. We can see that 256 is not a valid block size while 2048 and 51200 are.
4. If the disk platters rotate at 5400rpm, the time required for one complete rotation, which is the maximum rotational delay, is

$$\frac{1}{5400} \times 60 = 0.011 seconds$$

. The average rotational delay is half of the rotation time, 0.006 seconds.

5. The capacity of a track is 25K bytes. Since one track of data can be transferred per revolution, the data transfer rate is

$$\frac{25K}{0.011} = 2,250K bytes/second$$

Exercise 9.6 Consider again the disk specifications from Exercise 9.5, and suppose that a block size of 1024 bytes is chosen. Suppose that a file containing 100,000 records of 100 bytes each is to be stored on such a disk and that no record is allowed to span two blocks.

1. How many records fit onto a block?
2. How many blocks are required to store the entire file? If the file is arranged sequentially on the disk, how many surfaces are needed?
3. How many records of 100 bytes each can be stored using this disk?

4. If pages are stored sequentially on disk, with page 1 on block 1 of track 1, what page is stored on block 1 of track 1 on the next disk surface? How would your answer change if the disk were capable of reading and writing from all heads in parallel?

5. What time is required to read a file containing 100,000 records of 100 bytes each sequentially? Again, how would your answer change if the disk were capable of reading/writing from all heads in parallel (and the data was arranged optimally)?

6. What is the time required to read a file containing 100,000 records of 100 bytes each in a random order? To read a record, the block containing the record has to be fetched from disk. Assume that each block request incurs the average seek time and rotational delay.

Answer 9.6 Answer omitted.

Exercise 9.7 Explain what the buffer manager must do to process a read request for a page. What happens if the requested page is in the pool but not pinned?

Answer 9.7 When a page is requested the buffer manager does the following:

1. The buffer pool is checked to see if it contains the requested page. If the page is in the pool, skip to step 2. If the page is not in the pool, it is brought in as follows:
   (a) A frame is chosen for replacement, using the replacement policy.
   (b) If the frame chosen for replacement is dirty, it is *flushed* (the page it contains is written out to disk).
   (c) The requested page is read into the frame chosen for replacement.
2. The requested page is *pinned* (the *pin_count* of the chosen frame is incremented) and its address is returned to the requester.

Note that if the page is not pinned, it could be removed from buffer pool even if it is actually needed in main memory. *Pinning* a page prevents it from being removed from the pool.

Exercise 9.8 When does a buffer manager write a page to disk?

Answer 9.8 Answer omitted.

Exercise 9.9 What does it mean to say that a page is *pinned* in the buffer pool? Who is responsible for pinning pages? Who is responsible for unpinning pages?

Answer 9.9 1. *Pinning* a page means the *pin_count* of its frame is incremented. Pinning a page guarantees higher-level DBMS software that the page will not be removed from the buffer pool by the buffer manager. That is, another file page will not be read into the frame containing this page until it is unpinned by this requestor.

2. It is the buffer manager's responsibility to pin a page.

3. It is the responsibility of the requestor of that page to tell the buffer manager to unpin a page.

Exercise 9.10 When a page in the buffer pool is modified, how does the DBMS ensure that this change is propagated to the disk? (Explain the role of the buffer manager as well as the modifier of the page.)

Answer 9.10 Answer omitted.

Exercise 9.11 What happens if a page is requested when all pages in the buffer pool are dirty?

Answer 9.11 If there are some unpinned pages, the buffer manager chooses one by using a *replacement policy*, flushes this page, and then replaces it with the requested page.

If there are no unpinned pages, the buffer manager has to wait until an unpinned page is available (or signal an error condition to the page requestor).

Exercise 9.12 What is *sequential flooding* of the buffer pool?

Answer 9.12 Answer omitted.

Exercise 9.13 Name an important capability of a DBMS buffer manager that is not supported by a typical operating system's buffer manager.

Answer 9.13 1. Pinning a page to prevent it from being replaced.

2. Ability to explicitly force a single page to disk.

Exercise 9.14 Explain the term *prefetching*. Why is it important?

Answer 9.14 Answer omitted.

Exercise 9.15 Modern disks often have their own main memory caches, typically about 1 MB, and use this to prefetch pages. The rationale for this technique is the empirical observation that, if a disk page is requested by some (not necessarily database!) application, 80% of the time the next page is requested as well. So the disk gambles by reading ahead.

1. Give a nontechnical reason that a DBMS may not want to rely on prefetching controlled by the disk.
2. Explain the impact on the disk's cache of several queries running concurrently, each scanning a different file.
3. Is this problem addressed by the DBMS buffer manager prefetching pages? Explain.
4. Modern disks support *segmented caches*, with about four to six segments, each of which is used to cache pages from a different file. Does this technique help, with respect to the preceding problem? Given this technique, does it matter whether the DBMS buffer manager also does prefetching?

Answer 9.15 1. The pre-fetching done at the disk level varies widely across different drives and manufacturers, and pre-fetching is sufficiently important to a DBMS that one would like it to be independent of specific hardware support.

2. If there are many queries running concurrently, the request of a page from different queries can be interleaved. In the worst case, it cause the cache miss on every page request, even with disk pre-fetching.
3. If we have pre-fetching offered by DBMS buffer manager, the buffer manager can predict the reference pattern more accurately. In particular, a certain number of buffer frames can be allocated *per* active scan for pre-fetching purposes, and interleaved requests would not compete for the same frames.
4. *Segmented caches* can work in a similar fashion to allocating buffer frames for each active scan (as in the above answer). This helps to solve some of the concurrency problem, but will not be useful at all if more files are being accessed than the number of segments. In this case, the DBMS buffer manager should still prefer to do pre-fetching on its own to handle a larger number of files, and to predict more complicated access patterns.

Exercise 9.16 Describe two possible record formats. What are the trade-offs between them?

Answer 9.16 Answer omitted.

Exercise 9.17 Describe two possible page formats. What are the trade-offs between them?

Answer 9.17 Two possible page formats are: *consecutive slots* and *slot directory*

The consecutive slots organization is mostly used for fixed length record formats. It handles the deletion by using bitmaps or linked lists.

The slot directory organization maintains a directory of slots for each page, with a ⟨*record offset*, *record length*⟩ pair per slot.

The slot directory is an indirect way to get the offset of an entry. Because of this indirection, deletion is easy. It is accomplished by setting the length field to 0. And records can easily be moved around on the page without changing their external identifier.

Exercise 9.18 Consider the page format for variable-length records that uses a slot directory.

1. One approach to managing the slot directory is to use a maximum size (i.e., a maximum number of slots) and allocate the directory array when the page is created. Discuss the pros and cons of this approach with respect to the approach discussed in the text.
2. Suggest a modification to this page format that would allow us to sort records (according to the value in some field) without moving records and without changing the record ids.

Answer 9.18 Answer omitted.

Exercise 9.19 Consider the two internal organizations for heap files (using lists of pages and a directory of pages) discussed in the text.

1. Describe them briefly and explain the trade-offs. Which organization would you choose if records are variable in length?
2. Can you suggest a single page format to implement both internal file organizations?

Answer 9.19 1. The linked-list approach is a little simpler, but finding a page with sufficient free space for a new record (especially with variable length records) is harder. We have to essentially scan the list of pages until we find one with enough space, whereas the directory organization allows us to find such a page by simply scanning the directory, which is much smaller than the entire file. The directory organization is therefore better, especially with variable length records.

2. A page format with *previous* and *next* page pointers would help in both cases. Obviously, such a page format allows us to build the linked list organization; it is also useful for implementing the directory in the directory organization.

Exercise 9.20 Consider a list-based organization of the pages in a heap file in which two lists are maintained: a list of *all* pages in the file and a list of all pages with free space. In contrast, the list-based organization discussed in the text maintains a list of full pages and a list of pages with free space.

1. What are the trade-offs, if any? Is one of them clearly superior?
2. For each of these organizations, describe a suitable page format.

Answer 9.20 Answer omitted.

Exercise 9.21 Modern disk drives store more sectors on the outer tracks than the inner tracks. Since the rotation speed is constant, the sequential data transfer rate is also higher on the outer tracks. The seek time and rotational delay are unchanged. Considering this information, explain good strategies for placing files with the following kinds of access patterns:

1. Frequent, random accesses to a small file (e.g., catalog relations).
2. Sequential scans of a large file (e.g., selection from a relation with no index).
3. Random accesses to a large file via an index (e.g., selection from a relation via the index).
4. Sequential scans of a small file.

Answer 9.21

1. Place the file in the middle tracks. Sequential speed is not an issue due to the small size of the file, and the seek time is minimized by placing files in the center.
2. Place the file in the outer tracks. Sequential speed is most important and outer tracks maximize it.
3. Place the file and index on the inner tracks. The DBMS will alternately access pages of the index and of the file, and so the two should reside in close proximity to reduce seek times. By placing the file and the index on the inner tracks we also save valuable space on the faster (outer) tracks for other files that are accessed sequentially.
4. Place small files in the inner half of the disk. A scan of a small file is effectively random I/O because the cost is dominated by the cost of the initial seek to the beginning of the file.

Exercise 9.22 Why do frames in the buffer pool have a pin count instead of a pin flag?

Answer 9.22 Answer omitted.