Exercises: Disk and Buffer Management; File Organization

1. **Disk I/O**

   Exercise 9.2, 9.4, Ramakrishnan and Gehrke

2. **Page Formatting**

   Exercise 9.16, 9.17, 9.18, Ramakrishnan and Gehrke

3. **Column Stores**

   What are the differences between how row and column-store databases store records? What are the benefits and drawbacks of a column-store?

4. **Large records**

   In class, we assumed that each page could fit many records. What if we want to store large records (e.g., text documents or images) that cannot fit all in one page. How could we structure records and pages to accommodate this type of representation.

5. **Love/Hate Buffer Management**

   Theoretically, the best candidate page for replacement is the page that will be last requested in the future. Since implementing such policy requires a future predicting oracle, all buffer replacement policies try to approximate it one way or another. The LRU policy, for example, uses the past access pattern as an indication for the future. In this assignment you are to implement the *love/hate* replacement policy. The policy tries to enhance prediction of the future by relying on a hint from the upper levels about the page. The upper level user gives a hint to the buffer manager that the page is *loved* if it is more likely that the page will be needed in the future, and is *hated* if it is more likely that the page will not be needed. The policy should maintain an MRU list for the hated pages and an LRU list for the loved pages.

   A situation may arise when a page is both loved and hated at the same time by different processes. In this case, assume that "love conquers hate", meaning that once a page is indicated as loved it should remain loved.

   Answer the following questions in **1-2 sentences each**:

   (a) Briefly, why is LRU an appropriate policy for *loved* pages as opposed to MRU.

   (b) How would an upper-level process use this buffer policy to avoid *sequential flooding*? Your answer should identify (a) what type of access leads to sequential flooding and (b) what designation of love/hate the process would give when unpinning a page.

   (c) What data structure(s) would you used to keep track of loved pages? Under what condition would an item be added to the data structure?

   (d) Answer the same question for hated pages.

(e) Given your answers above, provide concise pseudocode for your replacement algorithm (i.e., `allocateFrame()`). You only need to identify the `FrameId` of the frame to replace (do not worry about handling dirty bits or resetting frames). Your code should only be a few lines long.

6. **Comparing Replacement Policies** Refer to the textbook to refresh your understanding of the LRU, Clock, and MRU strategies. Assume you are given a buffer manager with 4 frames (all empty to start). Below, I have given a series of page requests where each row indicates a time step and the page that is read from disk. Each page is read and then unpinned immediately before the next page is read. Simulate all three replacement policies, independently. For each, provide a list of pairs of each time a page is removed from the buffer pool by the policy. The pair should consist of the time step and letter indicating the page removed. For example, if during LRU, the 5th page request causes page A to replace page B and the 8th to replace A, your list will appear as: **LRU Policy: (5, B), (8,A)**. For each of the three policies, provide:

   - The list of replaced-page pairs described above
   - The state of all meta-data after time-step 8. This includes the buffer pool array of 4 pages with the letter of the page currently residing as well as the state of any data structures utilized. For the clock algorithm, this means you should show the refbit for each page and position of the clock hand.

Time Step	01	02	03	04	05	06	07	08
Page Read	A	B	C	D	A	A	B	E

Time Step	09	10	11	12	13	14	15	16
Page Read	F	A	G	B	A	B	C	G