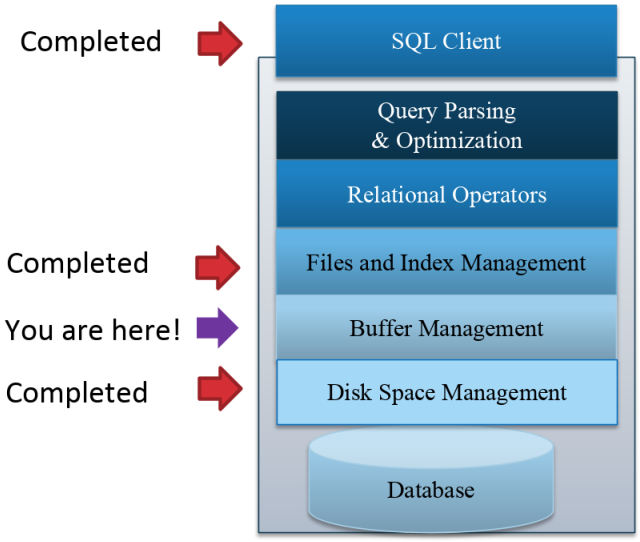
CS 186
Fall 2021

Buffer Management

1 Introduction

So far, we have discussed how disk space is managed at the lowest level of the database management system and how files and indexes are managed in our page-based database system. We will now explore the interface between these two levels on the DBMS - the buffer manager.



The buffer manager is responsible for managing pages in memory and processing page requests from the file and index manager. Remember, space on memory is limited, so we cannot afford to store all pages in the buffer pool. The buffer manager is responsible for the eviction policy, or choosing which pages to evict when space is filled up. When pages are evicted from memory or new pages are read in to memory, the buffer manager communicates with the disk space manager to perform the required disk operations.

2 Buffer Pool

Memory is converted into a buffer pool by partitioning the space into frames that pages can be placed in. A buffer frame can hold the same amount of data as a page can (so a page fits perfectly into a frame). To efficiently track frames, the buffer manager allocates additional space in memory for a metadata table.

Frame ID	Page ID	Dirty Bit	Pin Count
0	5	1	3
1	3	0	1
2	10	1	0
3			

The table tracks 4 pieces of information:

1. **Frame ID** that is uniquely associated with a memory address
2. **Page ID** for determining which page a frame currently contains
3. **Dirty Bit** for verifying whether or not a page has been modified
4. **Pin Count** for tracking the number of requestors currently using a page

3 Handling Page Requests

When pages are requested from the buffer manager and the page already exists within memory, the page's pin count is incremented and the page's memory address is returned.

If the page does not exist in the buffer pool and there is still space, the next empty frame is found and the page is read into that frame. The page's pin count is set to 1 and the page's memory address is returned. In the case where the page does not exist and there are no empty frames left, a replacement policy must be used to determine which page to evict.

The choice of replacement policy is heavily dependent on page access patterns and the optimal policy is chosen by counting page hits. A **page hit** is when a requested page can be found in memory without having to go to disk. Each **page miss** incurs an additional IO cost, so a good eviction policy is critical for performance. The **hit rate** for an access pattern is defined as # of page hits / (# of page hits + # of page misses) or more simply, # of page hits / # of page accesses.

Additionally, if the evicted page has the dirty bit set, the page is written to disk to ensure that updates are persisted. The dirty bit is set to 1 if and when a page is written to with updates in memory. The dirty bit is set to 0 once the page is written back to disk.

Once the requestor completes its workload, it is responsible for telling the buffer manager to decrement the pin count associated with pages that it previously used.

4 LRU Replacement and Clock Policy

A commonly used replacement policy is LRU (Least Recently Used). When new pages need to be read into a full buffer pool, the least recently used unpinned page which has pin count = 0 and the lowest value in the Last Used column is evicted. To track page usage, a last used column is added to the metadata table and measures the latest time at which a page's pin count is decremented.

Frame ID	Page ID	Dirty Bit	Pin Count	Last Used
0	5	1	3	20
1	3	0	1	32
2	10	1	0	40
3	6	0	0	25
4	1	0	1	15

Implementing LRU normally can be costly. The Clock policy provides an alternative implementation that efficiently approximates LRU using a ref bit (recently referenced) column in the metadata table and a clock hand variable to track the current frame in consideration.

Frame ID	Page ID	Dirty Bit	Pin Count	Ref Bit
0	5	1	3	1
1	3	0	1	1
2	10	1	0	1
3	6	0	0	1
4	1	0	1	0

Clock Hand
2

The Clock policy algorithm treats the metadata table as a circular list of frames. It sets the clock hand to the first unpinned frame upon start and sets the ref bit on each page's corresponding row to 1 when it is initially read into a frame. The policy works as follows when trying to evict:

- Iterate through frames within the table, skipping pinned pages and wrapping around to frame 0 upon reaching the end, until the first unpinned frame with ref bit = 0 is found.
- During each iteration, if the current frame's ref bit = 1, set the ref bit to 0 and move the clock hand to the next frame.
- Upon reaching a frame with ref bit = 0, evict the existing page (and write it to disk if the dirty bit is set; then set the dirty bit to 0), read in the new page, set the frame's ref bit to 1, and move the clock hand to the next frame.

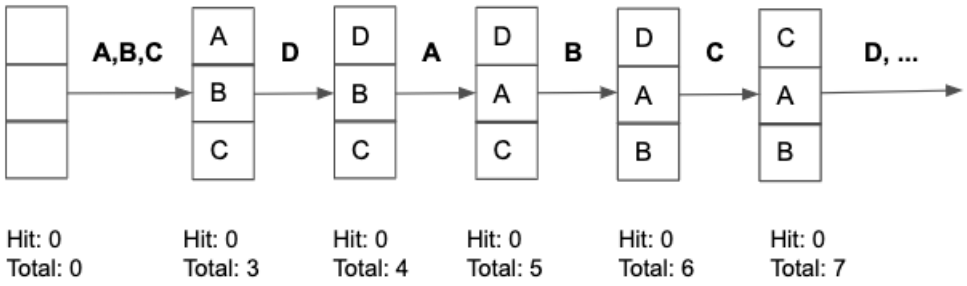
If accessing a page currently in the buffer pool, the clock policy sets the page's ref bit to 1 **without moving the clock hand**.

4.1 Sequential Scanning Performance - LRU

LRU performs well overall but performance suffers when a set of pages S, where $|S| >$ buffer pool size, are accessed multiple times repeatedly.

To highlight this point, consider a 3 frame buffer pool using LRU and having the access pattern:

A B C D A B C D A B C D A B C D



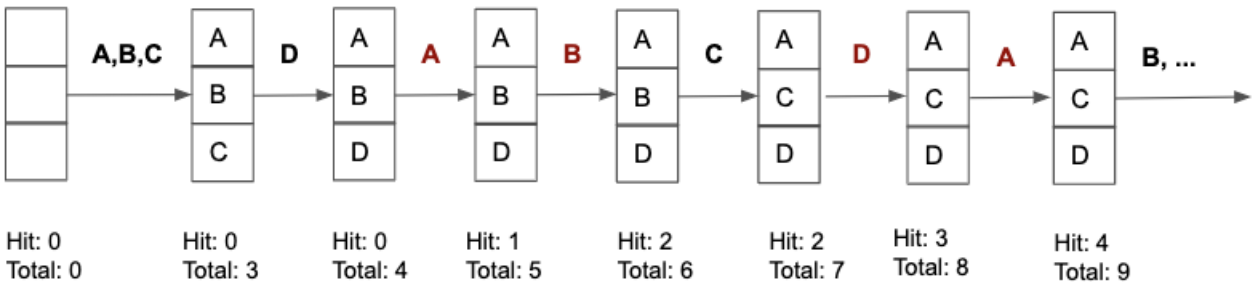
5 MRU Replacement

Another commonly used replacement policy is MRU (Most Recently Used). Instead of evicting the least recently used unpinned page, evict the most recently used unpinned page measured by when the page's pin count was last decremented.

5.1 Sequential Scanning Performance - MRU

At first it might seem counter-intuitive to use this policy but consider the scenario where a 3 frame buffer pool using MRU has the access pattern:

A B C D A B C D A B C D A B C D



Clearly, MRU far outperforms LRU in terms of page hit rate whenever a sequential flooding access pattern occurs.

6 Practice Questions

1. Determine whether each of the following statements is true or false.

   a. Each frame in the buffer pool is the size of a disk page.

   b. The buffer manager code (rather than the file/index management code) is responsible for turning on the dirty bit of a page.

   c. The dirty bit is used to track the popularity of the page.

   d. When using the LRU policy, the reference bits are used to give pages a "second chance" to stay in memory before they are replaced.

   e. A sequential scan of the file will have the same hit rate using either MRU or LRU (starting with an empty buffer pool) when the file is smaller than the buffer pool.

   f. The pin count of a page can only be decremented by the buffer manager.

2. For the following question, we are given an initially empty buffer pool with **4** buffer frames. For the access pattern:

   A B D D E A E C A B C A E

   a. What is the hit rate for LRU?

   b. What pages are in the buffer pool when LRU completes?

   c. What is the hit rate for MRU?

   d. What pages are in the buffer pool when MRU completes?

   e. What is the hit rate for Clock?

   f. What pages are in the buffer pool when Clock completes?

7 Solutions

1. a. True. A frame is defined as the size of a disk page.
   b. False. The requester of the page (file/index management code) sets the dirty page.
   c. False. The dirty bit is used to track whether the page has been modified before it's written back to disk. The pin is used to track popularity.
   d. False. Reference bits are not used for LRU policy. They are used for the Clock policy.
   e. True. Since we can fit all pages in memory, we do not need to evict pages during a sequential scan.
   f. False. The pin count is decremented by whoever requested the page to signify that they are done with it.

2. Detailed access patterns are in diagram below. In LRU and MRU, the number of rows corresponds to the number of buffer frames, and each column corresponds to a single access where characters represent misses and asterisks represent hits.
   a. 7/13
   b. A, C, B, E
   c. 7/13
   d. E, B, D, C
   e. 6/13
   f. C, A, B, E

LRU

A					*			*			*	
	B						C			*		
		D	*						B			
				E		*						*

MRU

A					*			*			*	E
	B								*	*		
		D	*									
				E		*	C			*		

Clock (after each access)

```
A: Miss      Clock: (A,1)(_,0)(_,0)(_,0)   Hand: 2
B: Miss      Clock: (A,1)(B,1)(_,0)(_,0)   Hand: 3
D: Miss      Clock: (A,1)(B,1)(D,1)(_,0)   Hand: 4
D: Hit       Clock: (A,1)(B,1)(D,1)(_,0)   Hand: 4
E: Miss      Clock: (A,1)(B,1)(D,1)(E,1)   Hand: 1
A: Hit       Clock: (A,1)(B,1)(D,1)(E,1)   Hand: 1
E: Hit       Clock: (A,1)(B,1)(D,1)(E,1)   Hand: 1
C: Miss      Clock: (C,1)(B,0)(D,0)(E,0)   Hand: 2
A: Miss      Clock: (C,1)(A,1)(D,0)(E,0)   Hand: 3
B: Miss      Clock: (C,1)(A,1)(B,1)(E,0)   Hand: 4
C: Hit       Clock: (C,1)(A,1)(B,1)(E,0)   Hand: 4
A: Hit       Clock: (C,1)(A,1)(B,1)(E,0)   Hand: 4
E: Hit       Clock: (C,1)(A,1)(B,1)(E,1)   Hand: 4
```