Data Management

Maurizio Lenzerini

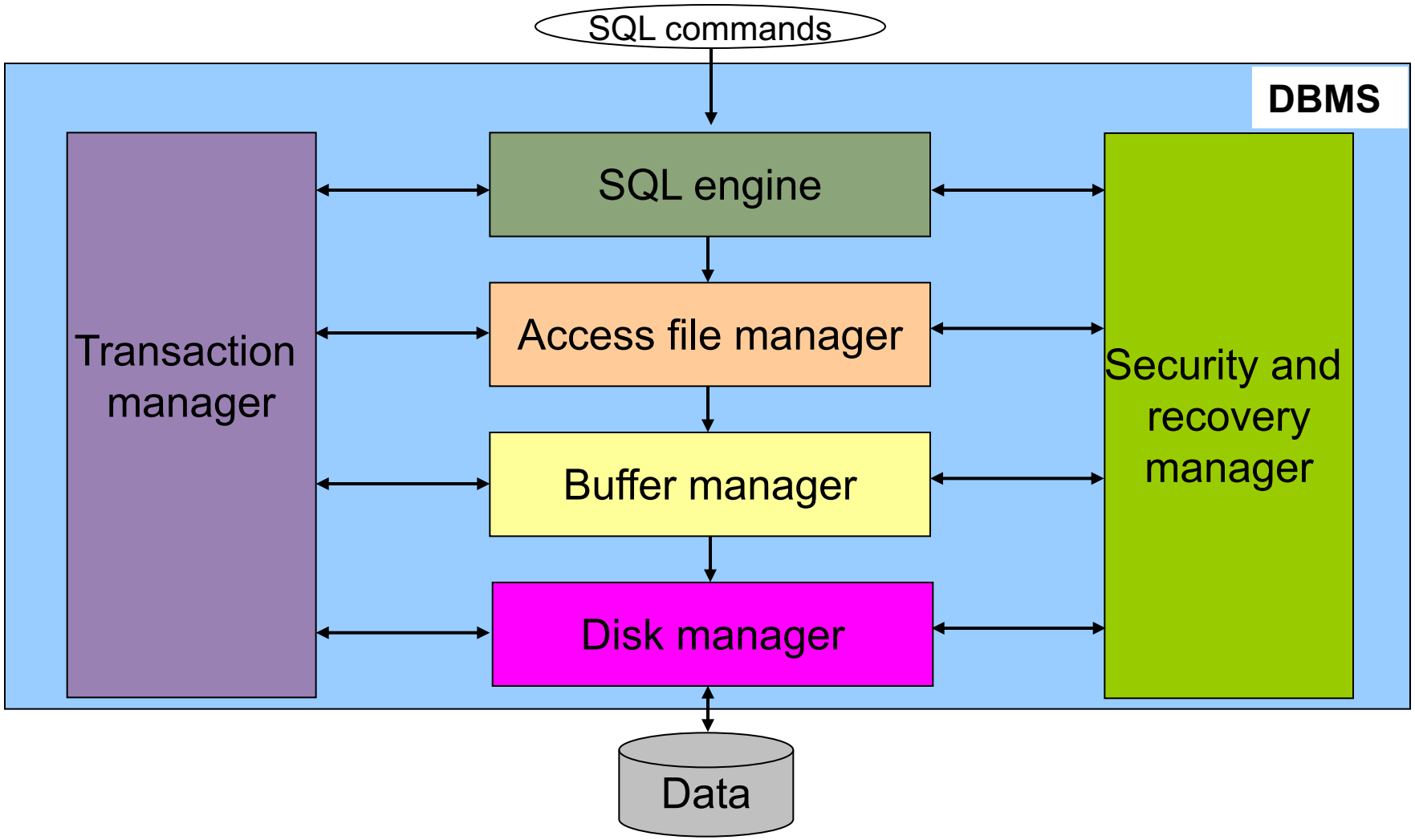
Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
Università di Roma "La Sapienza"

Academic Year 2020/2021

Part 2
Buffer management

http://www.dis.uniroma1.it/~lenzerin/index.html/?q=node/53

Architecture of a DBMS



2. Buffer management

The secondary storage

At the physical level, a data base is a set of database files, where each file is constituted by a set of pages, stored in physical blocks.

Using a page requires to bring it in main memory. The size of a block (and therefore of a page) is exactly the size of the portion of storage that can be transferred from secondary storage to main memory, and back from main memory to the secondary storage.

The buffer

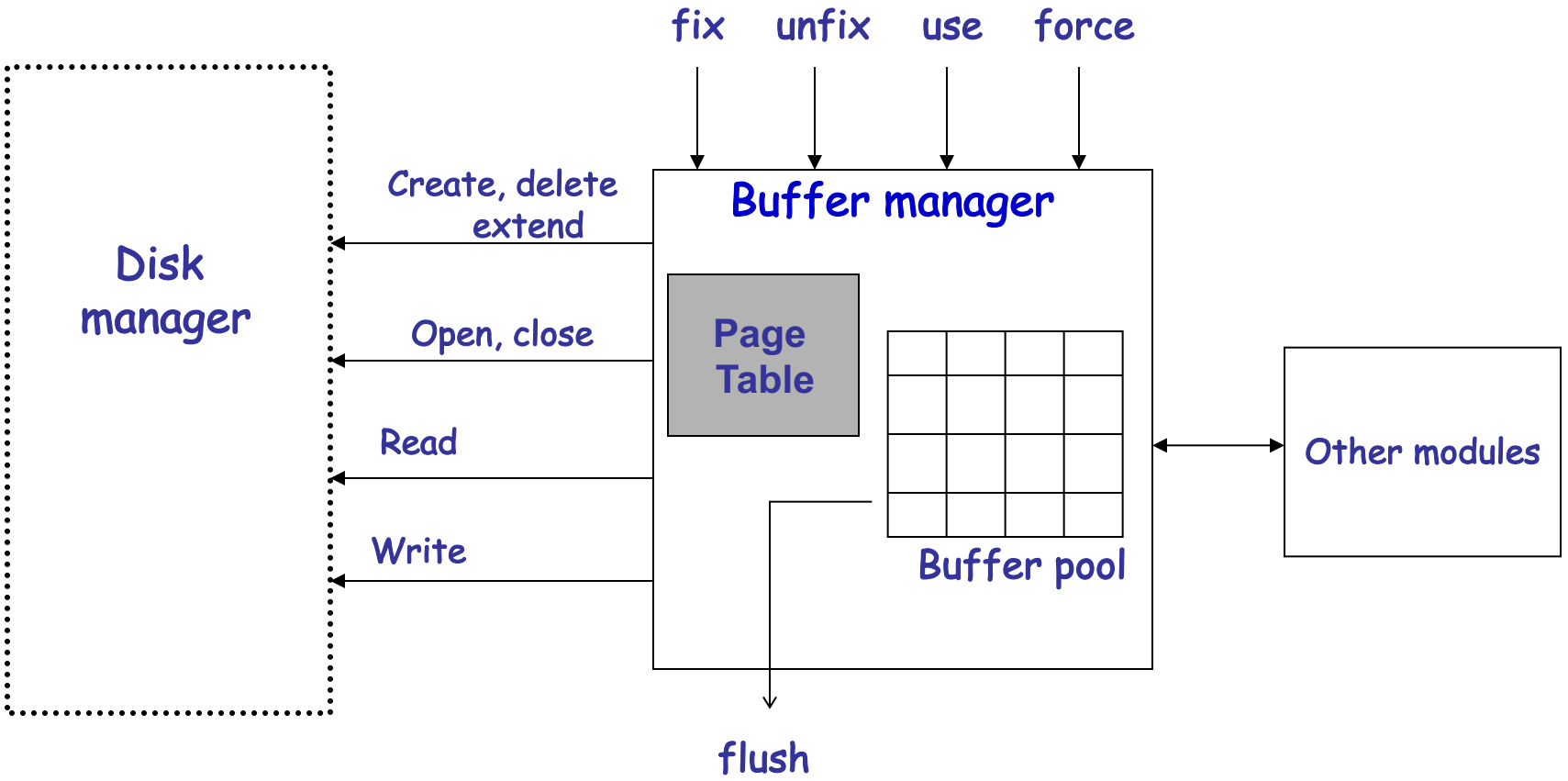
The buffer (also called buffer pool) is a non-persistent memory space which constitutes that portion of the main memory used by the DBMS to process the pages of the database.

The buffer manager is responsible of the transfer of the pages from the secondary storage to the buffer pool, and back from the buffer pool to the secondary storage.

The buffer pool is

- Shared by all transactions (i.e., all programs using the databases managed by the DBMS)
- Used by the system (e.g. by the recovery manager)

Architecture of buffer manager



The buffer pool

- The buffer pool is organized in “frames”, that are main memory pages, whose size is that of a block, where, as we said, a block is the transfer unit from/to the secondary storage. The typical size ranges from 2Kb to 64Kb.
- Since the buffer pool is in main memory, the management of their pages is more efficient (the cost of atomic operations is of the order of billionth of seconds) with respect to the cost of the management of secondary storage pages (thousandth of seconds)
- The buffer pool is managed with the same principles as the cache memory:
  - Locality principle
  - Block replacement strategy
  - Heuristics: 80% of applications access 20% of the pages

The buffer manager

The buffer manager uses the “Page Table” data structure, that associates to each page (denoted by its Page Identifier (PID)) appearing in the buffer, the frame number where the content of the page is stored.

It uses the following primitive operations:

- Fix: load a page
- Unfix: releases a page
- Use: uses a page in the buffer
- Force: Synchronous transfer to secondary storage
- Flush: Asynchronous transfer to secondary storage

The Fix operation

- An external module issues the ”Fix” operation in order to ask the buffer manager to load a specific page into a frame of the buffer

- Note that, for each frame, the buffer manager maintains:
  - The information about which page (if any) it contains (this information is provided by the Page Table)
  - pin-count: how many transactions use the page contained in the frame; at the beginning, it is set to 0.
  - dirty: a bit whose value indicates whether the content of of the frame has been modified (true) or not (false) from the last load; at the beginning, it is set to false.

The Fix operation

Suppose that module M issues a Fix operation that asks the buffer manager to load page P (denoted by its PID).

1. The buffer manager checks whether there is a frame F in the buffer pool already containing P
2. If yes, then it increments the pin-counter of F
3. If not, then
    1. In order to host page P in the buffer, a frame F' is chosen (if possible), according to a certain replacement policy
    2. If the dirty bit of frame F' is true (and therefore, a "steal" strategy is being used -- see later), then the page contained in F' is written back to secondary storage
    3. Page P is read from the secondary storage and is loaded in frame F'; the pin-count and the dirty bit of that frame are initialized to 0 and false, respectively
4. The address of the frame containing P is returned to module M

Replacement policy

The frame for the replacement (the "victim") is chosen among those with pin-count = 0. If no frame with pin-count=0 exists, then the request is placed in a queue, or the transaction is aborted. Otherwise, several policies are possible:

- LRU (least recently used): this is done through a queue containing the frames with pin-count=0
- Clock replacement:
  - We use a variable current pointing to frames in a circular fashion
  - Each frame has a bit referenced that tells us if the page contained in the frame has been used recently; referenced is set to true when pin-count is decremented to 0, to reflect the fact that the page is now a candidate for replacement, but it has been used recently
  - When we look for a frame for replacement, we analyze the frame pointed by current
  - If it has pin-count > 0, then we increment current and we repeat the procedure
  - If it has pin-count = 0, and referenced=true, we set referenced to false, to reflect the fact that time has now passed from the last usage of the corresponding page, we increment current and we repeat the procedure
  - If it has pin-count = 0 and referenced=false, then the frame is chosen for replacement

Steal/no-steal and force/no-force

- Steal/no steal:
  - Steal: we allow to chose the victim among those with dirty=true
  - No-steal: we do not allow to chose the victim among those with dirty=true
- Force/no force:
  - Force: all active pages of a transaction are written in secondary storage when the transaction commits (i.e., it ends its operations successfully)
  - No-force: the active pages of a transaction that has committed are written asynchronously in secondary storage through the flush operation

Comparison: steal/no-steal and force/no-force

- From the “recovery subsystem” point of view (activated when something goes wrong) point of view, no-steal and force are the most efficient strategies:
  - With no-steal, we can avoid the undo of transactions that have aborted
  - With force, we can avoid the redo of transactions that have committed, but whose effects have not been registered yet because of a failure
- However,
  - With no-steal, we are forced to keep many active pages in the buffer pool
  - With force, I/O operations tend to grow
- The steal, no-force strategy is the most common, because it enables a more efficient buffer management

The other operations

- Unfix:
  - The transaction certifies that it does not need the content of a specific frame anymore
  - The pin-counter of that frame is decremented
- Use:
  - The transaction modifies the content of a frame
  - The dirty bit is set to true and, if it is the first use of the frame by the transaction, pin-count is incremented
- Force: Synchronous (i.e., the transactions waits for the successful completion of the operation) transfer to secondary storage of the page contained in a frame
- Flush: Asynchronous (i.e., executed when the buffer manager is not busy) transfer to secondary storage of the pages used by a transaction

Exercise

Suppose the buffer of our DBMS contains 10 frames, where each frame at address *i* contains the page whose address in secondary storage is *i*, all the frames with an odd address have “pin-count”>0, all the frames with an even address have “pin-count”=0 and have “referenced”=true, and, finally, the value of “current” is 5. Describe all the actions performed by the buffer manager when the page of address 20 is asked with a fix operation, and illustrate the configuration of the buffer at the end of the execution of such operation.

Solution

When page P whose address is 20 is asked with a fix operation, since all the frames contain relevant pages, the buffer manager looks for a frame where P can be loaded. It accesses frame 5 (the one pointed by current) but all frames with odd addresses cannot be used to load P, because they have pin-count>0. Therefore, the buffer manager keeps scanning the frames, and since all the frames with even address have “pin-count”=0 and “referenced”=true, the buffer manager changes all such “reference” bits to false. After frame 10, the buffer manager will go back to 1, and eventually will return to frame whose address is 6. Now, such a frame F has “pin-count”=0 and “referenced”=false, and therefore, the buffer manager will load P in F, will register that the page in F is P, and will return 6 as the result of the fix operation. All other frames are not changed by the operation.