Buffer Manager and Buffer Management

Buffer Manager Overview

Buffer Manager is a component of a DBMS responsible for the following operations and services:

- Management and maintenance of *buffer space*: the main memory space designated to store data read from disk;
- Reading data from disk into buffer space;
- Flushing data from buffer space onto disk;
- Providing access to the relational data to the query execution layer of the DBMS;

Buffer space consists of M chunks of memory. The size of each chunk is equal to the size of a disk block. Each chunk (buffer) stores data from a single disk block.

The **Buffer Manager** has one key property:

> The **Buffer Manager** operates *asynchronously* w.r.t. other components of the DBMS[1]. When a Read() request is processed, the **Buffer Manager** ensures that by the end of the request the necessary data resides in the buffer. However, when a Write() request is processed by the **Buffer Manager**, the updated page does not immediately get written back to disk. Rather, it stays in the *buffer space* until a later time, when the **Buffer Manager** chooses to flush the page back to disk.

[1] The components include query execution layer, transaction scheduler and crash recovery module.

Buffer Management Strategies

When a new page needs to be brought into the buffer, the **Buffer Manager** needs to check whether available buffers exist. If all buffers are occupied, the **Buffer Manager** needs to select a buffer whose contents would be flushed back to disk.

Different *buffer management strategies* are possible. Here are some:

1. *LRU: Least Recently Used*. The rule is: *flush the block that has not been read or written for the longest time.* Buffer manager must maintain a table of timestamps for last access to each buffer. Each disk access makes an entry in the table.

2. *FIFO:First-In-First-Out*. The rule is: *flush the oldest block in the buffer space.* Table of timestamps needs to be maintained, but it is changed only when new page is read into the buffer space.

3. *The "Clock" Algorithm.* The rule is *rotate the slot that needs to be occupied.* Each buffer gets a counter, which is decreased by one, each time a check whether the buffer is available is made. When the counter goes to 0, the page is flushed. The next buffer to be checked is selected according to some predefined order.

Page Management in Main Memory

Reading disk pages. A **reading** operation results in loading the content of the designated page into some buffer page in the buffer space. This operation is instigated by the query execution layer, through the **transaction scheduler**, and the **Buffer Manager** executes it *synchronously*, i.e., after the **Buffer Manager** ends execution of this command, the desired page **will** reside in the buffer space.

Writing buffer pages. Write commands are also instigated by the query execution layer through the **transaction scheduler**, but unlike **read** commands, **write** commands are processed *asynchrnously*. The **Buffer Manager** executes **write** commands in two steps. On the first step, the designated page is marked as *ready* for being returned to disk, after which the **write** command proper successfully terminates. The second step is the operation of **flushing**, performed *asynchronously*.

Flushing buffer pages. Flushing is the operation of writing the content of a buffer page back to disk at the designated address. This operation is instigated by the **Buffer Manager** itself. The timing is determined by the **Buffer Manager** organization. When buffer space is full, **flushing** typically occurs when a new page needs to be moved from disk to the buffer space, to free up a buffer. **Flushing** may also occur on the background, if the overall DBMS workload experiences a temporary decrease. Additionally, it may be possible to force **flushing** from higher layers of the DBMS in some system architecutres.

Page Pinning/Unpinning. Generally speaking, when the **Buffer Manager** needs to select a page to flush to disk as part of its buffer management policy, any page in the buffer space will do. There are, sometimes situations, when a page stored in the buffer space is not ready to be returned to disk. **Pinning** is the operation of marking a buffer page to indicate to the **Buffer Manager** that it be flushed to disk at the moment, **Unpinning** is the operation of removing this mark.

When to pin/unpin pages?

You should **pin** a buffer page in the following situations:

- before writing any new content on the page (changing the data stored on it, or changing the values stored in the page header);
- when you know that the page will be needed for use later in the processing;
- when you know that the data as currently stored on the page is inconsistent (e.g., you need to update two fields of some record in the database: the page should remain pinned after you update first field but before you get to updating the second field).

A **pinned** page can be **unpinned** when

- the update of the page's content is complete, and the data on the page is in consistent state;
- the page is no longer needed (or, you do not have evidence that the page may be needed in the future);

Pinned pages present a strain on the *buffer management policies*, as they constrain the number of buffers that can be flushed. Because of it, careful management of **pinning/unpinning** should be employed.

There are two basic ways to keep track of **pinned** pages: internal and external. The *internal* way requires the system to reserve some space (1 byte is enough) in disk page headers. This byte is not used while the page is stored on disk, but when the page is in the buffer, it is used as the **pin flag**. The *external* way, means storing information about the **pinned/unpinned** status of disk pages in an external (to the buffer space) data structure, e.g. a lookup table.