Database Management System Implementation

Assignment No.2 – Group 22

The code pieces that were worked on or edited are:

- *bufmgr* class in *BufMgr.java*
- *FIFO* class in *FIFO.java*
- *LIFO* class in *LIFO.java*
- *LRUK* class in *LRKU.java*
- *history* class in LRUK.java

bufmgr class in *BufMgr.java*

- The constructor of this class is supposed to have a third integer argument called *lastRef.* The latter argument is related to the *LRUK* replacement algorithm and causes a conflict with the other pieces of code that we shouldn't edit. Therefore, it was not added to the *bufmgr* constructor.
- We added three new cases (for *FIFO,LIFO, and LRUK*) to the *bufmgr* class constructor and in the case of *LRUK*, *lastRef = 2* was inserted into its object creation.

FIFO class in *FIFO.java*

- As the other replacers, *FIFO* inherits from *Replacer* class.
- In this replacer, the *frames* array is like a queue where the index 0 is the head of the queue.
- When we want to pick a victim frame for a page that is not in the buffer
    - In the case where the buffer is not full we just choose and return the first empty frame in the array (the tail of the queue)
    - In the case where the buffer is full, we return the first frame from the head in which the page in it is unpinned.
- In the latter case we move all the other frames forward towards the head of the queue to fill the place of the chosen frame and then move the chosen from to the tail of the queue.

LIFO class in *LIFO.java*

- As the other replacers, *LIFO* inherits from *Replacer* class.
- In this replacer, the *frames* array is like a stack where the index 0 is the top of the stack.
- When we want to pick a victim frame for a page that is not in the buffer.
    - In the case where the buffer is not full we just choose and return the first empty frame in the array (the bottom of the stack)/empty frames have a negative value
    - In the case where the buffer is full, we return the first frame from the top in which the page in it is unpinned.

- In only these cases, we move all the other frames downwards away from the top of the stack to fill the place of the chosen frame and then move the chosen frame to the top of the stack at index 0.

history class in *LRUK.java*

- This class consists of all the parameters and the operations required to store, manipulate and utilize all the required history information.
- The actual history is represented as a hash map called *Hist* where the keys are the page IDs and the values are arrays representing the last K time instances in which the page was referenced.
- The other hash map called *Last* also has page IDs as keys and each value is the timestamp in which the corresponding page is last called for.
- *Last* may not be found in *Hist* due to the *Correlated_Reference_Period*
- The hash map helps organize the pages' history and is very efficient for this application.

LRUK class in *LRUK.java*

- The replacement code is based on the pseudo-code in the paper *"The LRU-K Page Replacement Algorithm For Database Disk Buffering"* By *Elizabeth J. O'Neil, Patrick E. O'Neill, & Gerhard Weikum.*
- In this class we have several additional properties which are:
    - *lastRef* which holds the value of K (as in last K accesses of a page)
    - *pageid* which is the ID of the page which is currently being called for. This allows us to build the history one page at a time.
    - *HIST* object which is an instantiation of the history class to keep track of the history parameters
    - *already_in_buffer* integer that keeps track of the status of the page having ID *pageid.*
- If the page is in the buffer, the *update(frameNo)* is called to update its *Hist* and *Last* information.
- In the process of picking a victim, if the buffer is not full, the first free frame is returned and the history for that page is generated.
- If the buffer is full, we look for a page *q* that has the oldest *Hist(q,K)* and the time passed since its last reference *List(q)* is greater than *Correlated_Reference_Period*
- Then the *update* function will be called to either generate a new *Hist* and *Last* blocks for the new page, or just update these two pieces of information in the case where the page was referenced in some point in history.