Prof. Dr. Stefan Brass
Institut für Informatik
MLU Halle-Wittenberg

December 1, 2017

Databases IIB: DBMS-Implementation
— Exercise Sheet 8 —

Part a) and c) to g) will be discussed in class, you only have to submit the Homeworks. This week, there are two deadlines: Part h) and i) should be submitted until December 6, 12:00, and the programming exercise j) until December 13. But please think about the questions in a) before the meeting.

Repetition Questions

a) What would you answer to the following questions in an oral exam?

- Name some storage characteristics and compare main memory with disks.
- What is a "logical block access", a "physical block access", a "cache hit" and a "cache miss"? How is the "hit ratio" defined?
- Suppose there were 1000 logical block accesses and 100 physical block accesses. What is the cache hit ratio?
- What are reasonable cache hit ratios for a normal OLTP database?
- Why does one often get reasonable cache hit ratios even if the buffer in main memory is much smaller than the data files on disk? E.g. if the main memory buffer is only 20% of the total database size, one would still expect a better hit ratio than 20%. Why?
- Suppose, one has a relatively large database (e.g. 60 GB), but also a large server machine with e.g. 128 GB RAM. What would be advantages and disadvantages of a standard DBMS with a cache for disk blocks compared to a system that first reads the entire data to memory?
- Why does a query run much faster if one executes it a second time (soon after the first time)?
- Suppose you want to compare the performance of two database management systems. You create the same database in both systems and measure the runtime of a few example queries. Why might the system vendors (or their salespeople) consider the result questionable?
- Explain the interface of a typical buffer manager.
- Explain the LRU replacement strategy. What do the letters "LRU" stand for?

- What is the problem with "sequential flooding" of the buffer? What is the solution in Oracle?
- What information might the upper layers of a DBMS have that they should pass to the buffer manager for improving the replacement strategy?
- What is the difference between "consistent gets" and "db block gets" in Oracle?
- How can one get information to determine the cache hit ratio in Oracle?
- Suppose you detect that the cache hit ratio is bad in your Oracle database. What can you do?
- What does the `CACHE` parameter of the `CREATE TABLE` do in Oracle?
- What is the purpose of the buffer pools `DEFAULT`, `KEEP` and `RECYCLE` in Oracle? How can they be used to improve the cache hit ratio?
- What is the main idea of the "Five Minute Rule"? Why might it be necessary to buy more disks although the overall disk space is sufficient? Why can buying more RAM solve the problem?

Some Classical Papers

b) Have a look at at least one of the following papers:

- Wolfgang Effelsberg and Theo Haerder:
  Principles of Database Buffer Management.
  ACM Transactions of Database Systems, Vol. 9, No. 4, Dezember 1984, pp. 560–595.
  [http://dx.doi.org/10.1145/1994.2022]
- Jim Gray and Gianfranco R. Putzolu:
  The 5 Minute Rule for Trading Memory for Disk Accesses and The 10 Byte Rule for Trading Memory for CPU Time.
  Proceedings of the ACM SIGMOD Conference, 1987, pp. 395–398.
  [http://dl.acm.org/citation.cfm?doid=38713.38755]
- Goetz Graefe:
  The Five-minute Rule: 20 Years Later and How Flash Memory Changes the Rules.
  ACM Queue, Vol 6, Issue 4, September 24, 2008.
  [http://queue.acm.org/detail.cfm?id=1413264]

You can access the PDFs of articles published by the ACM from IP-addresses of the university, because the university pays for using the ACM digital library.

In-Class Exercises

c) Which buffer pools does our Oracle database have? How large is the buffer pool, i.e. how much memory is used for the buffering of database blocks?

d) Create a table with the `BUFFER_POOL` and `CACHE` parameters. If you want, you can have a look at the `CREATE TABLE` reference:

- [https://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_7002.htm]

e) Check whether the initialization parameter "`TIMED_STATISTICS`" is true or false.

- [https://docs.oracle.com/cd/B28359_01/server.111/b28320/initparams245.htm]

You can set it locally for your session with the `ALTER SESSION` command:

- [https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_2012.htm]

It can be set globally with the `ALTER SYSTEM` command:

- [https://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_2013.htm]

After that, interesting statistics should be contained in `V$FILESTAT`.

f) As explained in Exercise Sheet 6, grant your normal user account the role `PLUSTRACE` (if you have not done that yet). Then enter "`SET AUTOTRACE ON STATISTICS`" into SQL*Plus. After that, execute some queries and look at the performance data that are shown. Compute the hit ratio from that. Notice that when you run the same query again, it will run much faster.

g) We can run `utlbstat.sql` at the start of the problem session and `utlestat.sql` after 30 minutes and then look at the generated report. Only one student should call `utlbstat`, since it stores the data in a global table. However, we can all look at the report generated by `utlestat.sql`.

These scripts are still available and interesting, but no longer the state of the art. Oracle has a library `STATSPACK` that permits to take many performance snapshots:

- [https://docs.oracle.com/cd/B10501_01/server.920/a96533/statspac.htm]
- [https://www.markusdba.de/?p=542]
- [http://www.orafaq.com/wiki/Statspack]

Actually, even newer is the "Automatic Workload Repository", however this is contained only in the Enterprise version (it is part of the "Diagnostics Pack"):

- [http://www.oracle.com/technetwork/database/manageability/]
  [diag-pack-ow09-133950.pdf]
- [https://docs.oracle.com/cd/E11882_01/server.112/e41573/autostat.htm]

Homework Exercises (Homework 8A, Deadline December 6)

h) Write an SQL query that computes and evaluates the hit ratio for Oracle:

$$\frac{\texttt{consistent gets} + \texttt{db block gets} - \texttt{physical reads}}{\texttt{consistent gets} + \texttt{db block gets}}$$

For example, you can access the value of the "physical reads" counter with the following query:

```
SELECT VALUE
FROM   V$SYSSTAT
WHERE  NAME = 'physical reads'
```

However, you have to write a single query that computes the hit ratio in percent (i.e. do not query the single counters and enter the data into a pocket calculator). Furthermore, your query should not only return the hit ratio, but in a second column one of the values "Good", "Medium", "Bad" depending on the value. Choose yourself reasonable boundaries for this evaluation.

i) Consider a buffer cache that operates with the LRU method, and has only four buffer frames. Suppose that the following blocks are accessed (pinned and immediately unpinned):

10, 12, 15, 20, 30, 12, 40, 15, 10, 12.

Which blocks are in the buffer at the end, and what was the hit ratio?

Homework Exercises (Homework 8B, Deadline December 13)

j) Define a class `buf_c` in C++ with the following components:

- A constant `BUF_MAXBUFS` for the maximal number of blocks that can be kept in memory (this is the number of buffer frames, i.e. the cache size).

  Every object of the class should represent one buffer frame. Thus every object would normally have to contain memory for one block (or a pointer to that memory). To simplify this first version, we only simulate the buffer, and do not really load blocks from a data file (that will be a future exercise).

  There can be at most `BUF_MAXBUFS` objects of this class. You can either generate objects as required up to this limit, or create immediately `BUF_MAXBUFS` objects (the simplest solution would be to use an array).

- A static method (class method) `init()` that you can use for initializations. It is guaranteed that the main program will call this method before any other method of the class is called. It will also be called only once.

- A static method `pin`, that is called with a file ID (a small non-negative number of type `int`) and a block number (a non-negative number of type `int`). It must return a pointer to a `buf_c`-object that contains the requested block. If the block is (in the simulation) already in a buffer frame, that object should be returned. Otherwise a new buffer frame must be allocated for the block. If there is no free buffer frame, and the `BUF_MAXBUFS` limit is reached (so that one cannot get more memory from the operating system), you must select a buffer with the LRU replacement strategy that can be reused. If that is impossible (too many `pin` without `unpin`), you should return the null pointer. If we would have a multi user system (that is too compilcated for these exercises), one could also wait for some time and try again.

- Object methods `file_id()` and `block_no()`, that can be used to query which block is contained in a buffer. As long as a block is pinned in the buffer, these values cannot change.

  If no block is pinned in the buffer, one could treat any access as a programming error (this is no requirement, but such tests could help to find bugs).

- A method `unpin()`, that unlocks the block in the buffer (i.e. it is not immediately removed, but it can be replaced by the LRU strategy).

  Please note that there can be several pins for the same block, and the buffer can only be reused if there are as many `unpin()`-calls as there were `pin()`-calls.

- Static methods `hits()` and `misses()`, that return the corresponding values, i.e. the total number of `pin()`-operations that found the block already in a memory buffer, and the total number of `pin()`-operations that had to "load" the block from disk (in this simulation or first version, the block is not actually read from disk).

- A static method `clear()` that marks all buffers as unused. This is at least helpful if one wants to do several tests without leaving the program. After calling `clear()`, the methods `hits()` and `misses()` should return `0` again.

In a real buffer manager one would need at least the following additional methods (not part of this exercise):

- A method `contents()`, that returns a pointer to the database block in the buffer (for blocks there would be a superclass `block_c` and subclasses depending on the contents, e.g. for blocks of a heap file or a B-tree).
- A method `set_modified()` that marks the contents of a buffer as modified. The buffer can only be reused after the block was written back to the data file.
- Support for setting a checkpoint that writes all modified blocks back to the data file. This works well only with several threads so that besides working on queries and updates the system can write blocks back from time to time (when it is idle). A minimal solution is a static method `checkpoint()` that writes all modified blocks back to the respective data file.
- Furthermore, one needs support for temporary data. Blocks with temporary data do not have to be written to a data file as long as their buffer is not needed otherwise. A minimal solution is to never remove temporary blocks from the buffer until they are deleted. However, this sets limits e.g. for sorting large data sets. A better solution is to use one or more files if an unpinned temporary block needs to be removed from the buffer cache.

Test your buffer management simulation with the block access sequence in the file

[`http://www.informatik.uni-halle.de/~brass/dbi17/refstring.txt`]

and a cache size (`BUF_MAXBUFS`) of 2000 blocks. How large is the hit ratio?

You can get a main program that reads the file and calls the methods `pin` and `unpin`, from the following directory:

[`http://www.informatik.uni-halle.de/~brass/dbi17/h8_buf/`]

The main program is `buftest.cpp`. The directory contains further files that might be useful, e.g. a `Makefile` and a first version of `buf.h` and `buf.cpp` that basically do not do anything but permit to compile the program. It is recommended to program more tests. You can extend `buftest.cpp` for this purpose.

Of course, your simulated buffer manager should work efficiently. Because `pin` is a frequent operation, you should try to find the block in the buffer cache fast. For instance, a hash table might be a good data structure. However, the homework will also be accepted if you do a simple linear search (you will not win a performance price with that).