Humboldt-Universität zu Berlin
Institut für Informatik
Prof. Dr. Ulf Leser
Arik Ermshaus

Berlin, the 14.11.2023

Tutorials for the lecture
Implementation of Databases (DBS2)

Exercise sheet 2

Submission: By Monday, on 04.12.2023, until 23:59 o'clock via Moodle. The exercise sheets are to be completed in groups of three (in exceptional cases two) students. Unless otherwise specified, the solutions must be submitted on separate **PDFs** via Moodle. All subtasks of a task must be uploaded in one PDF. It is sufficient for one person per group to submit the group's solution. For evaluation, the last uploaded version will be considered. Please include your **names**, your **CMS usernames**, and your **submission group (e.g., Gruppe 123)** from Moodle on all submissions. Name the uploaded PDF files according to the scheme: A<Task>-<Person1>-<Person2>-<Person3>.pdf, for example, A3-Musterfrau-Mustermann-Beispiel.pdf for Task 3 by Lisa Musterfrau, Peter Mustermann, and Karla Beispiel. The listing of names may be in any order. Please refer to the information in the Moodle course `https://hu.berlin/dbs223`.

Task 1 (Fixed and Variable-length Records) $2 + 2 + 3 = 7$ Points

Consider a database storing information about a collection of books in a library. There are 5 attributes that are always filled (e.g., title, author) and an additional 15 optional attributes that are not known or relevant for every book (e.g., awards won). Each of the optional attributes has the same probability p of being present.

Assume that the required attributes are stored in fields of 64 bytes each, and the optional attributes in fields of 16 bytes each. Based on this scenario, answer the following questions.

(a) Assume that the attributes are stored as "fixed-length" records. What is the size in bytes for a given record? *Hint:* Empty optional fields are filled with NULL values.

(b) Assume that the attributes are stored as "variable-length" records. The end of an attribute is terminated with a tag of 2 bytes. What is the expected size in bytes (dependent on p) for a given record? *Hint:* Empty optional fields are only represented by their tag.

(c) For which range of probabilities p should one favour fixed-length records?

Provide the way of calculations for each subtask.

Task 2 (Storing Relations in Blocks) 2 + 2 + 1 = 5 Points

Consider two relations, authors and books, where books are associated with authors via a foreign key. All tuples are stored as fixed-length, unspanned records. The authors relation has 5000 records, each consisting of 512 bytes. The books relation has 20,000 records, each one has 128 bytes.

The records are to be stored in blocks of size 4096 bytes, with each block having 64 bytes reserved for header information. Blocks are filled as much as possible. Answer the following questions, based on this information.

(a) How many blocks are required, if both relations are stored in separate groups of blocks? This means, a given block contains either author or book records, but not both.

(b) Assume, that books are distributed equally over all authors and that each book has only one author. How many blocks are required, if each author record is stored with its book records in the same block?

(c) Which scenario (subtask a or b) is preferable, if a query needs to output the entire author relation? Justify your choice.

Provide ways of calculations for subtask (a) and (b).

Task 3 (Buffer Manager) 5 + 7 + 6 = 18 Points

In Moodle, you find C++ and header files that in part implement basic record, block and caching data structures of a simple data base system. Your assignment is to complete this implementation to realize a basic buffer manager with a LRUn (Least Recently Unfixed) replacement strategy. To do so, you should implement the following functions:

(a) **Block::write_data**: This function stores the content of a block in a local file.

(b) **BufferManager::fix_block**: This function retrieves a block by its id from the cache or the file system and registers the fix. *Hint:* Blocks can be fixed multiple times. If the requested block is not already contained in the cache, it is emplaced in it. If the cache is full, an unfixed block is replaced according to the LRUn strategy. If the replacement is not possible, because no cached blocks are unfixed, a runtime error is thrown.

(c) **BufferManager::unfix_block** This function unfixes a block by its id. If the block is not present in the cache or it cannot be further unfixed, a runtime error is thrown.

Familiarise yourself with the codebase and check the Tutorial 2 slides for an example of the LRUn replacement strategy. To test your implementations, run the provided "main" method. Do not modify any of the provided function signatures or implementations. You are, however, free to implement additional helper functions or data structures.

Your code should rely solely on the C++ standard libraries; usage of third-party libraries is not permitted. Also, ensure that your code is executable on the gruenau2-6 system using the provided "CMakeLists.txt" file running the tests. Only hand-in the block and buffer manager C++ and header files.