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

Berlin, the 05.12.2023

Tutorials for the lecture
Implementation of Databases (DBS2)

Exercise sheet 3

Submission: By Monday, on 08.01.2024, 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 (Hierarchical Index-Sequential Files) 4 + 5 = 9 Points

Consider a relation $R(\underline{A}, B, C)$ that comprises 12 tuples, stored in blocks of a sequential file, sorted by the primary key $\underline{A}$. Within one data block, two tuples of R can be stored. The following figure shows the block distribution of the tuples of R in the sequential file, with only the value of primary key $\underline{A}$ specified for each tuple.

$(0,*,*)$	$(2,*,*)$	$(4,*,*)$	$(6,*,*)$	$(8,*,*)$	$(10,*,*)$
$(1,*,*)$	$(3,*,*)$	$(5,*,*)$	$(7,*,*)$	$(9,*,*)$	$(11,*,*)$

(a) Index the given data file with a two-level primary index $\mathcal{I}$ on $\underline{A}$, using a sparse index at each level. Within an index block, four key-pointer pairs can be stored.

(b) Annotate each block of the index structure and the sequential file with an unique identifier (e.g., a number). Then consider the following SQL queries and specify the identifiers of the blocks that must be read for the query execution *using the two-level primary index* $\mathcal{I}$.

(i) `SELECT * FROM R WHERE A = 6`

(ii) `SELECT * FROM R WHERE A > 7`

(iii) `SELECT COUNT(*) FROM R WHERE A = 8`

(iv) `SELECT COUNT(*) FROM R WHERE A = 9`

(v) `SELECT MIN(A) FROM R`

Hint: No blocks of R and no blocks of $\mathcal{I}$ are in main memory.

Task 2 (Hash Files) $2 + 2 + 2 = 6$ Points

In this task, we consider a relation $S(A, \underline{B}, C)$ that comprises 16384 fixed-length records, stored in blocks of a hash file. Each block can store 128 records and is completely filled. We can access the file with a predefined uniform hash function $h(\underline{B})$, using attribute $\underline{B}$ (a 8-bit unsigned integer), and a hash table H with 64 entries.

(a) What is the amount of records per bucket in the hash file?

(b) How many blocks have to be loaded to retrieve the records for a single value of $\underline{B}$. Assume that the search is unsuccessful and no records are found.

(c) Why is a hash file in general a bad choice of data structure for a range query? Provide an explanation.

Provide your ways of calculation for subtasks (a) – (b).

Task 3 (B+ Trees) $7 + 8 = 15$ Points

In Moodle, you find C++ and header files that in part implement a basic B+ tree data structure of a simple data base system. Your task is to complete the insert functionality of this implementation. Attributes are of type integer and duplicates are not allowed. Specifically, you should implement the following two methods for the B+ tree node object:

(a) **BPTreeNode::insert_record**: This function inserts an attribute with its associated record ID in the leaf node at the correct spot. If the node does not overflow, the method stops at this point. Otherwise, the node is split into two, attributes and record IDs are distributed accordingly and the newly created leaf node is returned, together with the median that separates it from the existing one. If a duplicate is tried to be inserted, an invalid argument exception is raised.

(b) **BPTreeNode::insert_value**: This function inserts a value with its left and right children IDs in the internal node at the correct spot. If the node does not overflow, the method stops at this point. Otherwise, the node is split into two, values and children IDs are distributed accordingly, parent node references are adapted and the newly created internal node is returned, together with the median that separates it from the existing one.

Familiarise yourself with the codebase and check the Tutorial 3 slides for examples of the B+ tree insert operation. 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 B+ tree C++ file.