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

Berlin, the 09.01.2024

Tutorials for the lecture
Implementation of Databases (DBS2)

Exercise sheet 4

Submission: By Monday, on 29.01.2024, 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 (Multi-dimensional Index Structures) — 2 + 2 + 2 = 6 Points

In this task you should analyze three query scenarios and determine what kind of multi-dimensional index structure is most suitable to reduce I/O access. You can choose from a composite index, a grid file or using no index at all, just scanning the data. Provide the attributes used in the index or for scanning and justify your choice. For the composite index, also specify and justify the order of used attributes.

(a) A city traffic monitoring system collects master data from various cameras across the city in a table. Each data point includes an external ID, the exact location of the camera (latitude and longitude), technical specifications (resolution, model number, manufactorer) and installation date. For a given camera, the system needs to frequently retrieve neighbouring ones in a partially open-ended area within the city, to analyze traffic patterns and congestion.

(b) An online bookstore has a large database containing thousands of recent books (year 2000 and onwards) covering 100 topics. The book table includes records with title, author, publication year, topic, and ISBN number. Users frequently search for books using a combination of both the topic and publication year or simply topics.

(c) A popular movie streaming service has a comprehensive database, with approximately 100 million records, cataloging every view of movies and TV shows in a table. Each record includes the movie/show ID, user ID, viewing date, duration of viewing, and viewer's age group. Once a month, the marketing team conducts extensive analyses where they sample at random 40-60% of data points with all attributes to understand viewing patterns, popular viewing times, and demographic preferences.

Task 2 (Query Execution Plans) **3 + 3 + 3 = 9 Points**

Consider the relation $R(\underline{A}, B, C, D)$ containing 1 million tuples. Assume that the attribute $\underline{A}$ is a primary key for R and its values of a range between 0 and 999,999. Each data block of the relation R accommodates 10 tuples and is completly filled. The tuples in R are unspannend and not sorted.

For each of the following queries (a to c), calculate the worst-case number of I/Os needed for each execution plan (i to iii); provide your ways of calculation. Then identify and state the execution plan that likely requires the fewest I/O operations to process the query.

Consider the following three execution plans:

(i) Scan table of relation R.

(ii) Use B^+ tree index structure to search for attributes $R.\underline{A}$ and load retrieved records. Tree height is $h = 3$ and every leaf node block contains exactly 100 values/tuple IDs.

(iii) Use direct block hashing to search for attributes $R.\underline{A}$ and load retrieved records. Hash blocks can store up to 100 values/tuple IDs. Each hash bucket consists of exactly two hash blocks (with each 50% fill degree).

The two listed data structures (in ii and iii) already exist and do not need to be created. All data, index and hash blocks are stored on the hard disk and cause I/O costs per query. Loading any block from R costs 1 I/O.

The three queries are:

(a) Return all tuples in R.

(b) Return all tuples in R, where $\underline{A} < 100$.

(c) Return all tuples in R, where $\underline{A} = 100$.

Task 3 (Query Execution) **8 + 7 = 15 Points**

In Moodle, you find C++ and header files that in part implement query execution operators of a simple database system. See the “Projection“ class for an example. It consists of ”open” and ”close” methods that prepare and conclude its data source, as well as a ”next” function that projects the given tuple. Your task is to implement the ”Selection” and ”Distinct” operators as pipelines, which implement just the same interface. Specifically, the “next“ methods should fulfill the following specifications:

(a) **Selection::next**: This function reads the next tuple from its data source. It compares the attribute, at the provided position with the given value and comparator. If the comparison evaluates true, the method returns the record. Otherwise, it reads the next tuple (if present) and repeats the process until it returns. If no record matches, the function returns a null pointer.

(b) **Distinct::next**: This function reads the next tuple from its data source. It checks with an internal data structure if the record has not been issued to its results before. In this case, the method returns the tuple. Otherwise, it reads the next record (if present) and repeats the process until it returns. If no tuples qualify, the function returns a null pointer.

Besides the “next“ methods, you are also tasked to correctly prepare the data source (or internal data structures) in the “open“ methods, and dispose them in the “close“ functions. If you use internal data structures, they must be based on the provided record and block architecture and be managed by the buffer manager. You must unfix all temporary fixed blocks at the end of a “next“ function and delete them in the “close“ method.

Familiarise yourself with the codebase and check the Tutorial 4 slides for query execution examples. 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 query execution header and C++ file.