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

Berlin, the 30.01.2024

Tutorials for the lecture
Implementation of Databases (DBS2)

# Exercise sheet 5

**Submission:** By Monday, on 12.02.2024, 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 (Query Optimization) — $5 \cdot 2 = 10$ Points

In the following tasks, you should work out different aspects of rule-based query optimization.

(i) Write the following query as an algebraic term:

```
SELECT R.a, S.c FROM R, S WHERE R.a = S.b AND S.c > 10
```

(ii) Draw one possible operator tree for the following query:

```
SELECT R.a, S.c, T.d FROM R, S, T WHERE R.a = S.b AND S.c = T.d AND T.d <= 20
```

(iii) Rewrite the following algebraic term, so that the number of used operators is mimimal: $\pi_{T.d,U.f}(\sigma_{T.d!=5}(\pi_{T.d,U.e,U.f}(\sigma_{T.d=U.e}(\pi_{T.d,U.e,U.f}(T \times U)))))$ *Hint:* Count the absolute amount of used operators to determine minimality.

(iv) Does a minimal representation of a query as an algebraic term nessecarily lead to minimal intermediate results in its execution? Provide an explanation.

(v) Optimize the algebraic term from subtask (iii) following the "rule-based optimizer". Note every used rule and rewrite the expression. *Hint:* See the lecture slides on "query optimization", page 45.

**Task 2 (Cost Estimation)** $5 \cdot 2 =$ **10 Points**

Consider the relations $R, S$ and $T$ as well as their statistics as shown in the following table. $|U|$ denotes the amount of tuples for a relation $U$ and $v(U, a)$ is its number of unique attribute values $U.a$. Assume that all values are uniformly distributed.

| $R(a, b)$ | $S(b, c)$ | $T(c, d)$ |
|---|---|---|
| $\|R\| = 300$ | $\|S\| = 500$ | $\|T\| = 900$ |
| $v(R, a) = 30$ | | |
| $v(R, b) = 60$ | $v(S, b) = 50$ | |
| | $v(S, c) = 100$ | $v(T, c) = 90$ |
| | | $v(T, d) = 60$ |

Unbiasdly estimate the result cardinality of the following algebraic terms. Provide your ways of calculation. Assume that all constants are part of the respective attribute values.

(i) $R \bowtie_{R.b=S.b} S \bowtie_{S.c=T.c} T$

(ii) $\sigma_{a=10}(R)$

(iii) $\sigma_{d>10}(T)$

(iv) $S \times T$

(v) $\sigma_{c>1 \wedge d=2}(T)$

**Task 3 (Joins)** **10 Points**

In Moodle, you find C++ and header files that in part implement query execution operators of a simple database system. See the “Table“, “Projection“, “Selection“ and “Distinct“ classes for examples. They consist of ”open” and ”close” methods that prepare and conclude data sources, as well as a ”next” function that forwards, transforms or filters a given tuple. Your task is to implement a ”Join” operator as a blocked execution. You can choose between a block nested loop, sort-merge or hash-based join. Specifically, the “open“ method should fulfill the following specifications:

(a) **Join::open**: This function prepares data sources and internal data structures. It reads all tuples from data source 1, joins them with the suitable tuples from data source 2, and saves them as temporary results in blocks. For a given tuple from data source 1, the method compares a candidate tuple from data source 2, with the provided comparator using the join attributes, at the given positions. If the comparison evaluates true, both tuples are concatenated and temporarily stored. Otherwise, the function checks other candidates.

Besides the “open“ method, you are also tasked to forward the joined tuples in the “next“ function, and dispose them in the “close“ method. Your internal data structures, to store the joined tuples, must be based on the provided record and block architecture and be managed by the buffer manager. Meta information, such as block ids, can be stored in class member variables. You must unfix all temporary fixed blocks at the end of the “open“ function and delete them in the “close“ method.

Familiarise yourself with the codebase and check the Tutorial 5 slides for join 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.