

---

EXAMINATION FOR  
“ARCHITECTURE AND IMPLEMENTATION OF DATABASE SYSTEMS”  
WINTER TERM 2009/2010,  
MARCH 2ND, 2010  
PROF. DR. RALF MÖLLER

---

**You are not allowed to write down solutions before the examination is started. Once the examination is officially ended, you are not allowed to write down solutions either. Violations of these rules count as attempts to cheat and will lead you to fail the exam.**

**Name:** \_\_\_\_\_

**Student id:** \_\_\_\_\_

**Course:** \_\_\_\_\_

**Signature:** \_\_\_\_\_

- a) **Please put your student identification as well as a passport/official id card on the table. We need to check these.**
- b) The exam will take **90 minutes**.
- c) The exam is **closed book**.
- d) The symbol “⊕” will give you hints on the **recommended time for solving a task**.
- e) We have more paper, should you need some, ask. Once you received additional sheets of paper, write down your name and student id.
- f) You are allowed to answer with notes (incomplete sentences).

20 ⊕  
20 P

# 1 Disks and storage

a) Nowadays, networks have higher bandwidths than traditional harddisks. Why can networks still be disadvantageous for storing database files in a classic database architecture?

b) *Least-Often-Used* page-replacement-algorithm: Why would one like to avoid *linear* space-complexity in implementations? What would be the key idea to obtain *constant* space-complexity? What is the drawback?

c) Write down pseudo code for the page-replacement algorithm *Least-Often-Used*. Here are some hints:

- Assume that you have to implement a procedure *access(p)*. After executing the procedure, the page *p* should be in memory.
- The choice of memory structures is up to you (just make sure that it is obvious to tell what is the in-memory structure in your pseudo-code). In your code you can use lists, sets, maps and other well defined collection classes (plus their standard operations).
- In your code you can use *LOAD p* and *SAVE p* to load pages from disk and write pages to disk, respectively. Pages should only be loaded if they are not in memory yet. Pages should only be saved if they are not needed anymore (with respect to the cache policy).
- Assume the number of pages in memory is limited 100.

Pseudo-Code:

15 ⌚

15 P

## 2 Indices

- a) Explain why B+-trees are rather good for single-dimension indices. Give an intuitive example (query) to explain why B+-trees don't directly work in the case of multi-dimensional indices. Name two approaches to make use of B+-trees for multi-dimensional indices (hint: linearization!).
- b) Name two index-structures for multi-dimensional indices. Explain their general idea and the major difference. What is the worst-case space overhead of null pointers for either of your two index-structures?

25 ⌚  
25 P

### 3 Sorting and Join

- a) Assume you have a memory buffer that allows you to store at most three pages at any given time, and each page of memory can contain at most two tuples. You wish to sort the following sequence of numbers: 33, 21, 12, 55, 6, 17, 94, 7, 34, 91, 16, 15, 14, 5, 47, 65, 80, 99, 8, 32. Perform a „vanilla“ external merge sort, by hand. No blocked I/O, no replacement sort, no double buffering. Show the result of each pass.

- b) Write down pseudo-code implementations for

A. Nested loop join:  $nljoin(R, S, p)$ :

B. Index nested loop join:  $inljoin(R, S, p)$ :

c) Sketch one hash- and one sort-based algorithm for the relational operator *intersection* in pseudo-code, i.e. implement the function *intersect*( $R, S$ ), where  $R$  and  $S$  are the input relations. The result ( $R \cap S$ ) should be stored in a variable named *result*.

A. Hash-based:

B. Sort-based:

15 ⌚  
15 P

## 4 Transactions

a) Consider the following schedule with 3 transactions (time increasing from left to right).

$r_3(x), w_1(x), r_2(y), w_1(y), r_2(x), w_3(x)$

b) Draw the serialization graph for the schedule.

c) Is the schedule conflict serializable? Explain why or why not.

d) Could the schedule have been generated by a scheduler using two-phase locking (2PL)?  
If so, proof it by injecting read/write lock/unlock operations in accordance with 2PL rules. If not, explain why.



20 ⊕  
20 P

## 6 Datalog

a) Explain the ideas of *Bottom-Up* vs. *Top-Down* approaches for query answering. Describe two situations in which either approach is more convenient to use.

b) Give an intuition why Datalog programs can be evaluated efficiently.

c) Is the number of deduced facts finite for Datalog programs? If yes, give a formula for a rough estimate of the bound, where  $p_i$  is the number of  $i$ -ary predicate symbols and  $c$  is the number of used constants in IDB+EDB.

