Database Management Systems
Written Examination

09.07.2009

First name		Last name	
Student number		Signature	

Instructions for Students

- Write your name, student number, and signature on the exam sheet.
- Write your name and student number on every solution sheet you hand in.
- This is a closed book exam: the only resources allowed are blank paper, pens, and your head. Use a pen, not a pencil.
- Write neatly and clearly. The clarity of your explanations affects your grade.
- You have 120 minutes for the exam.

Reserved for the Teacher

Exercise	Max. points	Points
1	20	
2	8	
3	20	
4	20	
5	12	
6	8	
7	12	
Total	100	

Exercise 1 (20 pt) Answer the following questions:

a. Mention two different techniques to optimize disk-block access?

b. What are two index structures that can efficiently handle multiple-key queries?

c. What are the two properties of an ideal hash function?

d. Assuming M memory blocks, what is the best way to use these blocks in the block nested loop join?

e. The non-leave nodes of a B^+-tree form a dense index on the leave nodes. Is that correct?

f. What are the 3 steps of query processing?

g. When is a schedule cascadeless?

h. What is stored in the lock table?

i. Does the two-phase-locking protocol ensure freedom from deadlocks?

j. For log-based recovery with immediate DB modifications: What actions are performed after a crash?

Exercise 2 (8 pt) The following table shows a file organization that uses variable-length records with the pointer method ("$\uparrow r_i$" denotes a pointer to record r_i, and $\perp$ denotes the end of a chain).

r_0	Jan	P1	400	$\uparrow r_2$
r_1	Joe	P3	350	
r_2		P2	500	$\perp$
r_3	Ann	P1	700	$\uparrow r_4$
r_4		P4	900	$\perp$

a. Show the file after the execution of the following steps:
   - Insert(Jan, P7, 800)
   - Insert(Ann, P2, 250)
   - Delete(Jan, P1, 400)

b. Transform the result of a) into a pointer representation that uses an anchor block and an overflow block.

c. What is the main disadvantage of the method in a) compared to the method in b)?

Exercise 3 (20 pt) Consider the following relation, which stores information about project assignments:

	Emp	Proj	Salary	Period
r_0	Joe	A	6	[1,6]
r_1	Joe	B	14	[1,12]
r_2	Ron	B	30	[4,24]
r_3	Ann	A	15	[7,18]
r_4	Jim	A	4	[7,12]
r_5	Ann	D	3	[10,12]
r_6	Lea	F	13	[12,24]
r_7	Ann	F	13	[13,24]
r_8	Jim	C	8	[13,18]
r_9	Max	B	7	[18,24]

Show the following index structures and file organisations (assume that the tuples are inserted in the order $r_0, r_1, \ldots$):

a. A primary B$^+$-tree index with $n = 3$ on *Proj* together with the data file. Show the structure after inserting $r_0, \ldots, r_4$ and after inserting all tuples.

b. A secondary index on *Emp* together with the data file.

c. A static hash file organisation on *Salary* using hash function $h(n) = n \mod 4$. Each bucket can hold at most 2 tuples.

Exercise 4 (20 pt) Consider a relation *emp* with schema $(ID, Name, Dept, Salary)$ and $|emp| = 1,000,000$. The size of the attribute values is: $ID = 4$ bytes, $Name = 50$ bytes, $Dept = 30$ bytes, $Salary = 6$ bytes. The *ID* values are equally distributed between 1 and $10,000,000$, and there are no two tuples with identical *ID*. A pointer occupies 6 bytes, and we assume a block size of $2,000$ bytes. Further, we assume seek time = 0.016 sec, latency = 0.016 sec, and transfer time = 0.001 sec.

a. Determine the number of blocks needed to store the relation *emp* if 10% of each data block are reserved for future insertions.

b. Assume a B$^+$-tree index on the attribute *ID*. Determine the number of blocks (= number of nodes) used for the B$^+$-tree if index blocks are filled up to 75%.

c. Consider the following two queries:

- Q1: $\sigma_{\texttt{ID}=1Mio}(emp)$
- Q2: $\sigma_{\texttt{ID}>7.5Mio}(emp)$

Determine the number of block IOs (index blocks and data blocks) and the execution time if the B$^+$-tree is a primary index.

d. The same as c), but assume that the B$^+$-tree is a secondary index.

Exercise 5 (12 pt) Let relations $r(A, B, C)$ and $s(C, D, E)$ have the following properties: r has 20,000 tuples, s has 45,000 tuples, 25 tuples of r fit on one block, and 30 tuples of s fit on one block.

Compute the costs of the following evaluation plans for $r \bowtie s$:

- Plan p1: Nested-loop join with r as outer relation
- Plan p2: Block nested-loop join with r as outer relation
- Plan p3: Merge join if r and s are not initially sorted

Exercise 6 (8 pt) Proof that the following expressions hold or do not hold:

a. $\sigma_\theta(E_1 - E_2) = \sigma_\theta(E_1) - E_2$

b. $\sigma_\theta(E_1 \cup E_2) = \sigma_\theta(E_1) \cup E_2$

Exercise 7 (12 pt) Given is the following schedule that involves transactions T_1 and T_2:

	T_1	T_2
1	read(A)	
2	write(A)	
3		read(A)
4		read(B)
5	read(B)	
6	write(B)	

Answer the following questions and explain your answers:

a. Is the schedule conflict serializable?

b. Is the schedule view serializable?

c. Is the schedule recoverable if both transactions commit immediately after the last operation?

d. Is the schedule possible under the timestamp protocoll?

Solution 1

a. Two of the following techniques: disk-arm-scheduling, appropriate file organization, use of write buffers, use of log disks.

b. Bitmap index and grid file index

c. Uniform and random

d. $M-2$ blocks for the outer relation, 1 block for the inner relation, 1 block for the output

e. No

f. (i) Parsing and transation, (ii) optimization, (iii) evaluation

g. If for each pair of transactions T_i and T_j such that T_j reads data previously written by T_i, the commit of T_i appears before the read of T_j.

h. The lock table stores granted locks and pending requests for locks.

i. No

j. Transaction T needs to be undone if the log contains a $\langle T, start\rangle$ record but not a $\langle T, commit\rangle$ record; T needs to be redone if the log contains both a $\langle T, start\rangle$ record and a $\langle T, commit\rangle$ record.

Solution 2

a. File after the 3 update operations:

r_0				
r_1	Joe	P3	350	
r_2	Jan	P2	500	$\uparrow r_5$
r_3	Ann	P1	700	$\uparrow r_4$
r_4		P4	900	$\uparrow r_6$
r_5		P7	800	$\perp$
r_6		P2	250	$\perp$

b. Pointer representation with anchor block and overflow block:

Anchor block:

r_0	Joe	P3	350	
r_1	Jan	P2	500	$\uparrow s_0$
r_2	Ann	P1	700	$\uparrow s_1$

Overflow block:

s_0	P7	800	$\perp$
s_1	P4	900	$\uparrow s_2$
s_2	P2	250	$\perp$

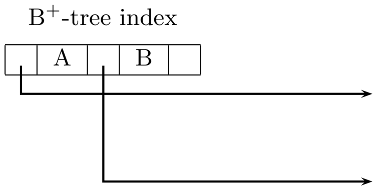
Note: This method (immediately) follows the pointer chains during the transformation, thus $(P7, 800, \perp)$ is the first overflow record. If the data records are scanned sequentially, the order of tuples in the overflow block is different.

c. Space is wasted (i.e., the Name attribute is empty) in all records except the first one in a chain.

Solution 3

a. Primary B$^+$-tree index on *Proj*

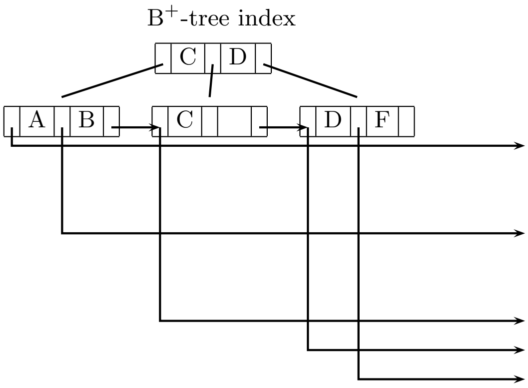
- after reading $r_0, \ldots, r_4$:



Data file

Emp	Proj	Salary	Period
Joe	A	6	[1,6]
Ann	A	15	[7,18]
Jim	A	4	[7,12]
Joe	B	14	[1,12]
Ron	B	30	[4,24]

- after reading all tuples

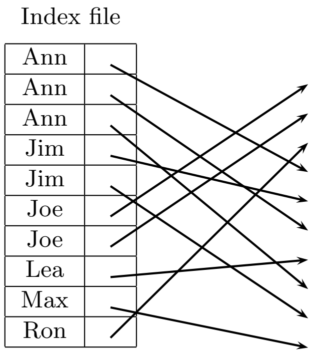


Data file

Emp	Proj	Salary	Period
Joe	A	6	[1,6]
Ann	A	15	[7,18]
Jim	A	4	[7,12]
Joe	B	14	[1,12]
Ron	B	30	[4,24]
Max	B	7	[18,24]
Jim	C	8	[13,18]
Ann	D	3	[10,12]
Lea	F	13	[12,24]
Ann	F	13	[13,24]

NOTE: For the data file we assume that tuples with identical *Proj* values are stored in the same order as in the original file, i.e., a new tuple is always stored at the end of a sequence of tuples with the same value.

b. Secondary index on *Emp* (with duplicate index entries)



Data file

Emp	Proj	Salary	Period
Joe	A	6	[1,6]
Joe	B	14	[1,12]
Ron	B	30	[4,24]
Ann	A	15	[7,18]
Jim	A	4	[7,12]
Ann	D	3	[10,12]
Lea	F	13	[12,24]
Ann	F	13	[13,24]
Jim	C	8	[13,18]
Max	B	7	[18,24]

c. Hash file organization

$h(4) = 0$; $h(8) = 0$; $h(13) = 1$; $h(6) = 2$; $h(14) = 2$; $h(30) = 2$; $h(3) = 3$; $h(7) = 3$; $h(15) = 3$;

	Bucket		Overflow buckets
Bucket 0:	(Jim, A, 4, [7,12]) (Jim, C, 8, [13,18])		
Bucket 1:	(Lea, F, 13, [12,24]) (Ann, F, 13, [13,24])		
Bucket 2:	(Joe, A, 6, [1,6]) (Joe, B, 14, [1,12])	⟶	(Ron, B, 30, [4,24])
Bucket 3:	(Ann, A, 15, [7,18]) (Ann, D, 3, [10,12])	⟶	(Max, B, 7, [18,24])

Solution 4

a. Data blocks:
$0.9 * 2,000 = 1,800$ bytes used in every data block
$\lfloor 1,800/(4 + 50 + 30 + 6) \rfloor = 20$ tuples/block
$\lceil 1,000,000/20 \rceil = 50,000$ data blocks needed for *emp*

b. Index blocks:
$0.75 * 2000 = 1,500$ bytes used in every index block
$\lfloor 1,500/(4 + 6) \rfloor = 150$ index entries/block
- level 3 (leaf nodes): $\lceil 1,000,000/150 \rceil = 6,667$ blocks
- level 2: $\lceil 6,667/150 \rceil = 45$ blocks
- level 1: $\lceil 45/150 \rceil = 1$ block
$\Rightarrow 6,713$ index blocks are needed

c. Primary index:
$Q1$:
3 index blocks (one at each level) + 1 data block
$\Rightarrow 3 + 1 = 4$ blocks are transfered in total

Execution time $= 4 * 0.033 = 0.132$ sec

$Q2$:
3 index blocks (one at each level) to locate the first tuple
$\lceil 250,000/20 \rceil = 12,500$ data blocks (on avg. $250,000$ tuples match, and all matching tuples are in contiguous blocks)
$\Rightarrow 3 + 12,500 = 12,503$ blocks are transfered in total

Execution time $= 12,503 * 0.033 = 412.6$ sec

d. Secondary index:
$Q1$: the same as for primary index

$Q2$:
On avg. $250,000$ data tuples match
3 index blocks (one at each level) to locate the first tuple
$\lceil 250,000/150 \rceil = 1667$ index (leaf) nodes/blocks to locate the other tuples
$250,000$ data blocks to retrieve (in the worst case each matching tuple might be in a different block)
$\Rightarrow 3 + 1,667 + 250,000 = 251,670$ blocks are transfered in total
(a full scan of the data relation without using the index would be more efficient)

Execution time $= 251,670 * 0.033 = 8,305.1$ sec

Solution 5
Tuples of r: $n_r = 20,000$
Tuples of s: $n_s = 45,000$
Blocks required for r: $b_r = \lceil 20,000/25 \rceil = 800$ blocks
Blocks required for s: $b_s = \lceil 45,000/30 \rceil = 1500$ blocks

a. $cost(p1) = b_r + n_r * b_s = 800 + 20,000 * 1,500 = 30,000,800$

b. $cost(p2) = b_r + b_r * b_s = 800 + 800 * 1,500 = 1,200,800$

c. Composed of cost of joining + cost of sorting (using external sort-merge)
$cost(p3) = b_r + b_s + cost(sorting)$

$cost$(sorting r) $= b_r(2\lceil log_{M-1}(b_r/M)\rceil + 1)$, where M is the number of buffer blocks. We have to add to this cost the cost of the output, i.e., b_r block transfers.

$cost$(sorting r) $= 800 * (2\lceil log_{M-1}(800/M)\rceil + 2)$
$cost$(sorting s) $= 1,500 * (2\lceil log_{M-1}(1,500/M)\rceil + 2)$

Total cost: $cost(p3) = 800 + 1,500 + 800 * (2\lceil log_{M-1}(800/M)\rceil + 2) + 1,500 * (2\lceil log_{M-1}(1,500/M)\rceil + 2)$

Solution 6

a. The equivalence holds. Proof by showing containment in both directions:
$\Rightarrow$: Assume $\exists t \in \sigma_\theta(E_1 - E_2)$; then t satisfies θ and $t \in E_1$ and $t \notin E_2$; therefore $t \in \sigma_\theta(E_1)$; since $t \notin E_2$, we have also $t \in \sigma_\theta(E_1) - E_2$.
$\Leftarrow$: Assume $t \in \sigma_\theta(E_1) - E_2$; then t satisfies θ and $t \in E_1$ and $t \notin E_2$; therefore $t \in E_1 - E_2$ and, since t satisfies θ, we have also $t \in \sigma_\theta(E_1 - E_2)$.

b. The equivalence does not hold. Proof by counter-example:
Assume schema (A, B) for E_1 and E_2, instances $E_1 = \{(a, 1)\}$ and $E_2 = \{(b, 1)\}$, and let θ be the condition $A =' a'$.
Then on the right-hand side we get $E_1 \cup E_2 = \{(a, 1), (b, 1)\}$ and $\sigma_{A='a'}(E_1 \cup E_2) = \{(a, 1)\}$.
On the left-hand side we get $\sigma_{A='a'}(E_1) \cup E_2 = \{(a, 1)\} \cup \{(b, 1)\} = \{(a, 1), (b, 1)\}$, which is different from the result of the left-hand side.

Solution 7

a. No. For both possible serial schedules, $\langle T_1, T_2\rangle$ and $\langle T_2, T_1\rangle$, we get either a conflict with $write(A) - read(A)$ or with $write(B) - read(B)$.

b. No.
In the serial schedule $\langle T_1, T_2\rangle$, the following rule is violated for data item B: For each data item Q, if transaction T_i reads the inital value of Q in schedule S, then T_i must in scheule S' also read the initial value of Q.
In the serial schedule $\langle T_2, T_1\rangle$, the following rule is violated for data item A: For each data item Q, if transaction T_i reads data item Q in schedule S and the value was produced by T_j, then T_i must in scheule S' also read the value of Q that was produced by T_j.

c. No. T_1 might fail after T_2 already committed (and T_2 used A which was produced by T_1).

d. No. Assume $ts(T_1) = 1$ and $ts(T_2) = 2$. Then T_2 sets the read timestamp of B to $R - ts(B) = 2$. When T_1 wants to write B, we have $ts(T_1) < R - ts(B)$, thus the write operation is rejected and T_1 is rolled back.
If $ts(T_2) = 1$ and $ts(T_1) = 2$ the same situation appears with the data item A.