Database Management Systems
Written Examination

25.09.2008

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	10	
3	20	
4	20	
5	10	
6	10	
7	10	
Total	100	

Exercise 1 (20 pt) Answer the following questions:

a. What are the three performance measures of hard disks?

b. What is the main problem with sequential file organisation?

c. How must the data be stored that binary search can be applied to evaluate a selection query?

d. What index is preferrable for range queries: primary index or secondary index? Explain your answer.

e. What is specified in a query plan (evaluation plan)?

f. Describe two alternative ways to evaluate the complex join $r \bowtie_{\theta_1 \vee \theta_2 \vee \cdots \vee \theta_n} s$

g. Which strategy for the evaluation of complex expressions is more efficient: materialization or pipelining?

h. When is a schedule cascadeless?

i. What is stored in the lock table?

j. Consider log-based recovery with immediate DB modifications and the following log file: $\langle T_0, start \rangle$, $\langle T_0, A, 1000, 950 \rangle$, $\langle T_0, B, 2000, 1950 \rangle$. What actions are performed if the system crashes in this situation?

Exercise 2 (10 pt) Given is the following table with project assignments:

Employee	Project	Hours
Jan	P1	800
Ann	P1	250
Jan	P2	400
Jan	P3	500
Jan	P4	900
Joe	P3	350

a. Show two different ways of a file organization using the reserved space method (variable-length records).

b. Assume the following memory requirements for the attribute values: Name = 20 Bytes, Project = 10 Bytes, Hours = 4 Bytes. Calculate the disk space used by the two different solutions in a).

c. How much disk space is wasted (unused) by the two solutions in a)?

Exercise 3 (20 pt) Consider the following relation r:

	Name	*Course*	*Grade*
r_0	Tom	ITP	30
r_1	Tom	DMS	21
r_2	Aron	CSA	18
r_3	Ann	OS	18
r_4	Ann	DMS	25
r_5	Nick	ITP	27
r_6	Nick	DSA	23
r_7	Nick	IDB	26
r_8	Sue	ITP	28
r_9	Sue	CSA	19

Show the following index structures and file organisations:

a. A primary dense B^+-tree index on *Course*. Assume $n = 3$ for the B^+-tree. The tuples are inserted in the order $r_0, r_1, r_2, \ldots, r_9$. Show the tree after inserting $r_0 - r_4$ and after inserting all tuples.

b. An extensible hash table on *Grade* with the hash function $h(n) = n \mod 8$. Each bucket holds at most 2 tuples. The tuples are inserted in the order $r_0, r_1, r_2, \ldots, r_9$. Show the structure after inserting $r_0 - r_4$ and after inserting all tuples.

c. Assume that we map the grades to an alternative system with grades A–E as follows: A = [30], B = [29, 28, 27], C = [26, 25, 24, 23, 22], D = [21, 20, 19], E = [18]. Create a bitmap index that efficiently supports queries that retrieve all tuples with a grade that corresponds to A, B, C, D, or E.

d. Assume that you have a B^+-tree index on *Course* and a (separate) B^+-tree index on *Grade*. Then consider the following query:

```
SELECT * FROM r WHERE Course = 'ITP' AND Grade = 30
```

Describe 3 different evaluation strategies for this query that take advantage of the two indexes (independent of the specific tuples in the above table).

Exercise 4 (20 pt) Assume a relation `prod(pid, category, price, ...)` with 600.000 tuples, where each tuple is 100 Bytes. The product ID `pid` is a key and is equally distributed between 1 and 3.000.000. The block size is 2.000 Bytes.

a. Consider a B^+-tree index on the product ID `pid`, where the `pid` requires 4 Bytes and a pointer requires 6 Bytes; a tree node occupies an entire block. Determine the minimal and maximal number of blocks used for the tree.

b. Consider the B^+-tree from a.) with the minimal number of blocks and assume that it is a primary index. Describe the evaluation of the following queries and determine the number of IOs (data blocks + index blocks):

```
Q1:SELECT * FROM prod WHERE pid BETWEEN 10000 AND 20000
Q2:SELECT CNT(*) FROM prod WHERE pid BETWEEN 10000 AND 20000
```

c. Repeat c.) but assume the B^+-tree to be a secondary index.

d. Consider to apply external sort-merge to sort the relation `prod`. The number of blocks that fit in the buffer is 40. Briefly describe the steps of the algorithm, indicate the number of sorted runs at each step, and determine the total number of block transfers.

Exercise 5 (10 pt) Assume two relations $r(A, C)$ and $s(B, D)$ and the following relational algebra expression:

$$\sigma_{A<10 \wedge B>100 \wedge A+B<200}(r \times s)$$

a. Transform this selection statement into a more efficient expression by applying some equivalence rules. Explain your choice and why the new expression is more efficient.

b. Assume that you can create one single-attribute index on either relation r or s to improve the evaluation of the expression obtained in a.). Which index (type, relation, attribute) would you create? Motivate your choice and briefly describe the evaluation strategy with this index.

Exercise 6 (10 pt) Given are two schedules S_1, S_2 over transactions T_1, T_2, T_3:

S1:

T_1	T_2	T_3
	read(A)	
read(B)		
	write(A)	
		read(A)
write(B)		
		write(A)
	read(B)	
	write(B)	

S2:

T_1	T_2	T_3
	read(A)	
read(B)		
	write(A)	
	read(B)	
		read(A)
write(B)		
		write(A)
	write(B)	

a. Show the conflict graphs of S_1 and S_2.

b. Are S_1 and S_2 conflict serializable? (explain your answer)

c. Are S_1 and S_2 view serializable? (explain your answer)

Exercise 7 (10 pt) Consider the following two transactions:

T_1: read(A);
read(B);
if A=0 **then** B := B + 1;
write(B).

T_2: read(B);
read(A);
if B=0 **then** A := A + 1;
write(A).

a. Add lock and unlock instructions to transactions T_1 and T_2, so that they observe the two-phase locking protocol.

b. Show a concurrent schedule of T_1 and T_2 that results in a deadlock? Show also the evolution of the wait-for graph.

c. For the schedule in b.) what happens under the wait-die deadlock prevention protocol?

Solution 1

a. Access time, data-transfer rate, and mean time to failure.

b. To maintain the physical order as records are inserted and deleted.

c. The data must be stored on contiguous blocks and sorted on the attribute in the selection condition.

d. Primary index. Use the index to locate the first data item, then scan the data file. With a secondary index, the index is needed to locate every matching record.

e. Query plan defines what algorithm is used for each operation, in what order the operations are exectued, and whether materialized evaluation or piplined evaluation is used.

f. (1) Use nested loop or block nested loop join and test the entire condition for each pair of tuples.
(2) Compute the union of the individual joins: $r \bowtie_{\theta_1} s \cup \cdots \cup r \bowtie_{\theta_n} s$

g. Pipelining.

h. 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.

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

j. $undo(T_0)$, i.e., A is restored to 1000, and B is restored to 2000.

Solution 2

a. File organization with reserved space method for variable-length records:
– Solution I: One record for each person

0	Jan	P1	800	P2	400	P3	500	P4	900
1	Ann	P1	250	⊥	⊥	⊥	⊥	⊥	⊥
2	Joe	P3	350	⊥	⊥	⊥	⊥	⊥	⊥

– Solution II: One record for each project

0	P1	Jan	800	Ann	250
1	P2	Jan	400	⊥	⊥
2	P3	Jan	500	Joe	350
3	P4	Jan	900	⊥	⊥

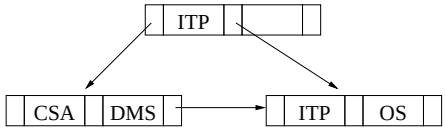
b. Memory requirements:
– Solution I: 1 record = (20 + 4 x 14) = 76 Bytes, total = 3 x 76 = 228 Bytes
– Solution II: 1 record = (10 + 2 x 24) = 58 Bytes, total = 4 x 58 = 232 Bytes

c. Unused disk space:
– Solution I: 6 x 14 = 84 Bytes
– Solution I: 2 x 24 = 48 Bytes

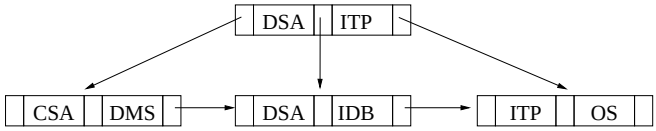
Solution 3

a. B$^+$-tree index

– after inserting $r_0 - r_4$:



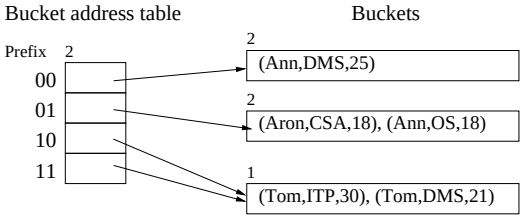
– after inserting all tuples:



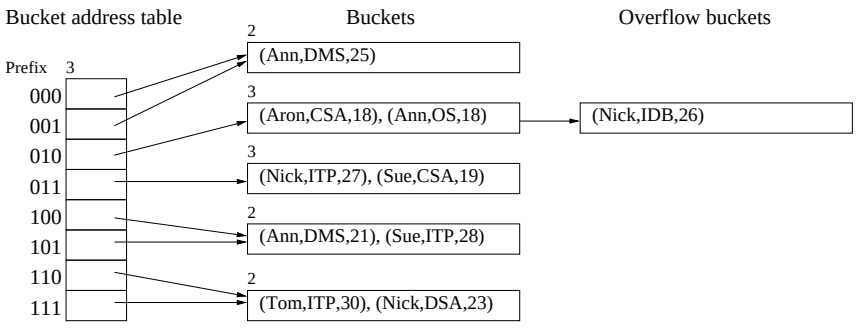
b. Extensible hash file organization (TODO):

- The hash function gives:
  $h(18) = 010$, $h(19) = 011$, $h(21) = 101$, $h(23) = 111$, $h(25) = 001$,
  $h(26) = 010$, $h(27) = 011$, $h(28) = 100$, $h(30) = 110$
- Overflow buckets are used, if a bucket is already full.

– after inserting $r_0 - r_4$:



– after inserting all tuples:



c. Bitmap index for *Grade*:

A: `[1 0 0 0 0 0 0 0 0 0]`
B: `[0 0 0 0 0 1 0 0 1 0]`
C: `[0 0 0 0 1 0 1 1 0 0]`
D: `[0 1 0 0 0 0 0 0 0 1]`
E: `[0 0 1 1 0 0 0 0 0 0]`

d. The 3 evaluation strategies are:

1. Use index on *Course* to find all tuples with *Course* = 'ITA'; then test for *Grade* = 30.
2. Use index on *Grade* to find all tuples with *Grade* = 30; then test for *Course* = 'ITA'.
3. Use index on *Course* to find pointers to all records with a *Course* = 'ITA'. Similarly, use index on *Grade* to find pointers to all records with a *Grade* = 30. Take the intersection of the two pointer sets.

Solution 4

a. Minimal number of index blocks when tree nodes are completely filled
$\lfloor 2.000/(4+6) \rfloor = 200$ index entries/block
- leaf nodes: $\lceil 600.000/200 \rceil = 3.000$ blocks
- level $n-1$: $\lceil 3.000/200 \rceil = 15$ blocks
- level $n-2$: $\lceil 15/200 \rceil = 1$ block
$\Rightarrow$ at least 3.016 index blocks are required

Maximal number of index blocks when tree nodes are only half full
$\lfloor 1.000/(4+6) \rfloor = 100$ index entries/block
- leaf nodes: $\lceil 600.000/100 \rceil = 6,000$ blocks
- level $n-1$: $\lceil 6.000/100 \rceil = 60$ blocks
- level $n-2$: $\lceil 60/100 \rceil = 1$ block
$\Rightarrow$ at most 6.061 index blocks are required

b. Average distance between `pid` values: $3.000.000/600.000 = 5$
$\Rightarrow Q1$ and $Q2$ retrieve $(20.000 - 10.000)/5 = 2.000$ tuples on average.

Data tuples/block: $\lceil 2.000/100 \rceil = 20$

Q1: Traverse the tree once to get the block of the first matching tuple, then scan the data blocks for the other tuples.
– Block IOs: 3 index nodes + $\lceil 2.000/20 \rceil = 100$ data blocks $\Rightarrow$ 103 total IOs

Q2: Traverse the tree once to get the leaf node with the first matching search-key, then scan the leave nodes for the other matching keys. The tuples are not needed to evaluate this query!
– Block IOs: 3 index nodes + $2.000/200 = 10$ index leaf nodes $\Rightarrow$ 13 IOs

c. Q1: Traverse the tree once to get the leaf node with the first matching search-key, then follow the leaf nodes for the other matching search-keys. For each matching search-key, follow the data pointer and retrieve the tuple.
– Block IOs: $3+10 = 13$ index nodes as in $Q2$ above; 2.000 data blocks (worst case, each tuple on separate block); $\Rightarrow$ 2.013 IOs in total

Q2: The same as in b.)

d. External sort-merge:
We have 20 data tuples/block;
Size of relation: $\lceil 600.000/20 \rceil = 30.000$ blocks
40 blocks fit into the buffer

Step 1: Create initial sorted runs by filling the buffer with 40 blocks of tuples and performing in-memory sort; each sorted run is of size 40 blocks (= 400 tuples).
– Number of sorted runs: $\lceil 30.000/40 \rceil = 750$

Step 2 (Merge pass 1): Merge runs by merging contiguous groups of 39 sorted runs, thus reducing the number of runs by a factor of 39.
– Number of sorted runs: $\lceil 750/39 \rceil = 20$

Step 3 (Merge pass 2): Perform merge as in step 2.
– Number of sorted runs: $\lceil 20/39 \rceil = 1$

Total number of block transfers:
– In step 1 and step 2 all blocks are read from and written back to disk $\Rightarrow 2 * (2 * 30.000) = 120.000$ IOs
– In step 3 we do not count the final output $\Rightarrow 1 * 30.000 = 30.000$ IOs
$\Rightarrow$ 150.000 IOs in total

Solution 5

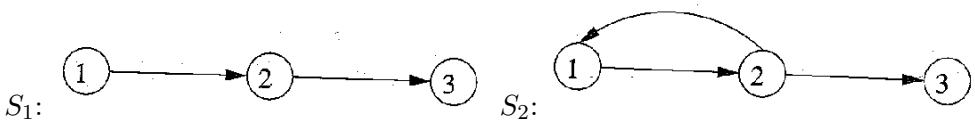
a. First, since condition $A < 10$ refers only to relation r, it can be pushed down to r. Similar, $B > 100$ can be pushed down to s, and we get $\sigma_{A+B<200}(\sigma_{A<10}(r) \times \sigma_{B>100}(s))$, which produces a smaller Cartesian product.

   Second, the condition $A + B < 200$ can be pushed down to transform the Cartesian product into a join. The final expression is then: $\sigma_{A<10}(r) \bowtie_{A+B<200} \sigma_{B>100}(s)$

b. Create an ordered index on attribute B of relation s. The index can then be used to locate the first tuple with $B > 100$ and then continue to scan the relation. For the condition on relation r is not useful, since the relation is anyway scanned from the beginning.

Solution 6

a. Conflict graphs



b. S_1 is conflict serializable, since the conflict graph contains no cycles. It is equivalent to the serial schedule $\langle T_1, T_2, T_3 \rangle$.
   S_2 is not conflict serializable, since there is a cycle in the conflict graph.

c. Since S_1 is conflict serializable, it is also view serializable.
   S_2 is not view serializable, since both T_1 and T_2 read the initial value of B and modify B. Hence, it is impossible to create a serial schedule, where both T_1 and T_2 read the initial value of B.

Solution 7

a. Lock and unlock instructions:

T_1:		T_2:	
	lock-S(A);		lock-S(B);
	read(A);		read(B);
	lock-X(B);		lock-X(A);
	read(B);		read(A);
	if A=0 **then** B := B + 1;		**if** B=0 **then** A := A + 1;
	write(B).		write(A).
	unlock(A);		unlock(B);
	unlock(B);		unlock(A);

b. The following schedule results in a deadlock at step 6:

	T_1	T_2	Wait-for graph
1	lock-S(A);		
2		lock-S(B);	
3		read(B);	
4	read(A)		
5	lock-X(B)		$T_1 \longrightarrow T_2$ (T_1 waits for T_2)
6		lock-X(B);	$T_1 \rightleftarrows T_2$ (T_1 waits for T_2 and vice versa)

c. Wait-die deadlock prevention protocol: We assume that T_1 is the older transaction and T_2 is the younger transaction. Then at step 6, T_2 (the younger transaction) will not wait for T_1 (the older transaction) to release the lock. Instead, T_2 is rolled back, and the lock on B is released. T_1 can now continue.