Data Management – AA 2013/14 – exam of 21/2/2014

Compito A

Problem 1
Suppose relations `R(A,B,C)` and `S(D,E,F)` are stored in 500 pages and 2000 pages, respectively, each without duplicates. Suppose also that we have 550 buffer frames available in main memory. Which is the algorithm you would use for computing the set union of `R` and `S` with a minimal number of page accesses? And which is the cost of such algorithm in terms of page accesses?

Solution 1
Since the buffer has 550 frames free, we can use the one-pass algorithm: we read `R` into 500 buffer frames free, we write all pages of `R` into the result, and then we read `S` one page at a time, and for each tuple, we check whether the tuple is already in the buffer frames. If yes, we ignore the tuple, otherwise, we write the tuple in the page devoted to the result (when such a page is full, we write it into the result). The number of page accesses (as usual, ignoring the pages used to write to result) is simply 500 + 2000 = 2.500.

Problem 2
Consider the relation `CONCERT(band,year,city,cost,people)` that stores information about concerts, with the band that gave the concert, the year when the concert has been held, the city where the concert has been held, the cost of the concert, and the number of people that attended the concert. The relation occupies 300.000 pages, each of 800 KB. We assume that all fields in any record have the same size of 10 KB, independently of the field type. There is a sparse B^+-tree index on `CONCERT` with search key $\langle$`band,year,city`$\rangle$, using alternative 2. Consider the query

```
select band,city from CONCERT;
```

that asks for the band and the city of all the concerts, and tell which algorithm you would use for executing the query, and how many page accesses such algorithm needs for computing the answer.

Solution 2
Since the index is sparse, we cannot use the index for answering the query, because we know that we have only one data entry for each page, and not for each value of the search key in the data records. Therefore, the only way to compute the result of the query is to use the algorithm for projection (possibly with duplicates), which is a one-pass algorithm. So, the number of page accesses is 300.000 (as usual, we ignore the cost of writing the result).
Comment:
Just as an observation, let us analyze the case of dense index also. In this case, since each page has size 800 KB, and the size of each field in every record is 10 KB, independently of the field type, we conclude that in every page we have space for 80 values. Since the index uses alternative 2, and every search key value has 3 fields, we know that every data entry has 4 fields (the three values of the search key, plus the pointer to the data file), and therefore every leaf of the index has room for 80/4 = 20 data entries. Taking into account the 67% occupancy rule, this means that every leaf of the index has 13 data entries. Since we are considering the case of dense index, we have one data entry for each tuple of the relation. How many tuples does the relation have? Since every tuple occupies 50KB, in every page we 800/50=16 tuples, and therefore, we have 300.000 * 16 = 4.800.00 tuples in the relation. Now, 4.800.000/13 = 369.230 is the number of leaves of the tree. Since the query asks for the band and the city of all the concerts, the search key contains band and city, and the index is now dense, we could in principle answer the query simply by an index-only scan. More precisely, we could use the one-pass algorithm computing the projection of the relation stored in the leaves of the index. However, the cost is 369.230 page accesses (as usual, we ignore the cost of writing the result), which is worse than just scanning the data file.

Problem 3

A "transaction with single-ending-write" has a single write operation, and such write operation is the final action of the transaction. Let S be a schedule all of whose transactions are "transactions with single-ending-write". Prove or disprove each of the following two properties.

- S is in the class of 2PL schedules.
- We can insert in S the "commit" operations of the various transactions appearing in S in such a way that the resulting schedule is strict.

Solution 3
We disprove that if S is a schedule all of whose transactions are transactions with single-ending-write, then S is a 2PL schedule, by showing a counterexample. It is easy to see that the schedule

$$r_1(x)\, r_2(y)\, w_2(x)\, w_1(y)$$

is a schedule all of whose transactions are transactions with single-ending-write, but is not a 2PL schedule. Indeed, for $w_2(x)$ to be executed, transaction 1 should release the lock on x, and for being 2PL, transaction 1 should get the lock on y before $u_1(x)$, but this implies that transaction 2 should release the shared lock on y before acquiring the lock on x, contradicting the 2PL rule.

We prove that if S is a schedule all of whose transactions are transactions with single-ending-write, then we can insert in S the "commit" operations of the various transactions appearing in S in such a way that the resulting schedule is strict. Indeed, consider the schedule S' obtained from S by adding, for each write action $w_i(z)$, the commit operation c_i of transaction T_i just after the action $w_i(z)$. Suppose S' is not strict. There are two cases:

1. transaction T_i reads from T_j, i.e., we have $r_i(x)$ reading from $w_j(x)$, and c_i is not between $w_j(x)$ and $r_j(x)$. But this contradicts the fact that c_i is just after $w_i(x)$;

2. transaction T_i writes on T_j, i.e., we have $w_i(x)$ writing on $w_j(x)$, and c_i is not between $w_j(x)$ and $w_i(x)$. But this contradicts the fact that c_i is just after $w_i(x)$.

So, we have proved that assuming S' is not strict leads to a contradiction, implying that S' is strict.

Problem 4
Consider the following schedule

$$S = r_3(z)\, r_1(z)\, w_2(y)\, w_4(x)\, w_3(z)\, w_3(y)\, r_1(x)\, w_2(x)$$

4.1 Tell whether S is view serializable or not, explaining the answer in detail. If the answer is yes, then tell if there is a single action that can be added to S in such a way that the resulting schedule is no longer view serializable. If the answer is no, then tell if there is a single action that can be deleted from S in such a way that the resulting schedule is view serializable.

4.2 Tell whether S is conflict serializable or not, explaining the answer in detail.

4.3 Tell whether S is accepted by the 2PL scheduler with exclusive and shared locks. If the answer is yes, then show the schedule obtained from S by adding suitable lock and unlock commands. If the answer is no, then explain the answer.

4.4 Tell whether S is ACR, whether it is strict, and whether it is rigorous, explaining the answer in detail.

Solution 4

4.1 S is view-serializable: the serial schedule T_4, T_1, T_2, T_3 is view-equivalent to S. There is a single action that can be added to S in such a way that the resulting schedule is no longer view serializable: for example, if we add $r_2(z)$ at the end of the schedule, then the resulting schedule S' is no longer view serializable, because now T_2 reads from T_3 (implying that in every serial schedule view equivalent to S', T_3 preceeds T_2), and in S the final write on y is $w_3(y)$ (implying that in every serial schedule view equivalent to S', T_2 preceeds T_3).

4.2 S is conflict serializable because the precedence graph associated to S is acyclic.

4.3 S is not accepted by the 2PL scheduler with exclusive and shared locks: in order to execute $w_3(z)$, T_1 must issue the command $u_1(z)$ before $w_3(z)$, and because of $r_1(x)$, T_1 should issue the command $sl_1(x)$ before $u_1(z)$; in order to execute $w_3(y)$, T_2 must issue the command $u_2(y)$ before $w_3(y)$, and because of $w_2(x)$, T_2 must issue the command $xl_2(x)$ before $u_2(y)$. However, $xl_2(x)$ is incompatible with $sl_1(x)$.

4.4 S is ACR, because T_1 reads from T_4, but T_4 can commit before $r_1(x)$. S is not strict, because T_3 writes on T_2, but T_2 cannot commit before $w_3(y)$. Obviously, since S is not strict, S is not rigorous.

Problem 5

Describe in detail the multi-pass sort-merge algorithm for sorting a relation in secondary storage. Which is the complexity of the algorithm in terms of page accesses?

Solution 5

See the slides of the course.