Data Management (A.A. 2024/25) – exam of 11/07/2025
Solutions

Problem 1
Consider a schedule S with both shared and exclusive locks, that is legal and is such that each of its transactions enjoys the following properties: (i) it is well-formed, (ii) it has all the lock operations before the commit action, and (iii) it has all the unlock operations after the commit action.

1.1 Provide the definition of view serializable schedule and prove or disprove that S is view serializable.

1.2 Provide the definition of strict schedule, and prove or disprove that S is strict.

Solution 1

1.1 *Definition:* A schedule S is view-serializable if there exists as serial schedule on the same set of transactions as S that is view-equivalent (same READ-FROM relation and same FINAL-WRITE set) to S.

It is immediate to verify that a schedule enjoying all the three properties $(i) - (iii)$ described above is a 2PL schedule, becuase no transaction gets a lock after an unlock. Since every 2PL scheule is conflict serializable and every conflict serializable schedule is view serializable, the thesis follows.

1.2 *Definition:* A schedule S is strict if no transaction reads from or writes on another transaction that has not committed yet.

It is immediate to verify that a schedule enjoying all the three properties $(i) - (iii)$ described above is strict, because whenever a transaction gets a lock, the transaction that released the lock has done so after the commit action. It follows that no transaction can read from or write on a transaction that has not committed yet, and the thesis follows.

Problem 2
Consider a system with processors $P_0, P_1, \ldots, P_{10}$, where (i) P_0 has 10 buffer frames and stores the relation `Exam(student,course,mark)` with 1.000 pages, (ii) each of the processors $P_1, \ldots, P_5$ have 100 buffer frames, and (iii) each of the processors $P_6, \ldots, P_{10}$ have 20 buffer frames. Consider the query

```
select student,avg(mark)
from Exam
group by student
```

and answer the following questions:

2.1 Which algorithm would you use for computing the result of the query if you could use only processor P_0, and which is the cost of such algorithm in terms of number of page accesses?

2.2 Which algorithm would you use for computing the result of the query if you could use processors $P_1, \ldots, P_{10}$ for a parallel execution, and which is the cost of such algorithm in terms of both number of page accesses and elapsed time?

Solution 2

2.1 If we can use only processor P_0, then we can use the multipass algorithm based on sorting. Since $10^3 = 1000$ and $10^4 \geq 1.000$, we can surely do it in 4 passes, with a cost of $(2 \times 4 - 1) \times 1.000 = 7.000$.

Actually, we can even do better, if we observe that for each tuple of `Exam` we only need to store the value of `student` and the value of `mark`. This implies that the number of pages required for `Exam` during the algorithm after the first pass (reading the relation) is $1.000/3 \times 2 = 667$ and that 3 passes suffice, with a total cost of 1.000 (cost of reading the relation) + $4 \times 667 = 3.668$.

2.2 If we can use the processors $P_1, \ldots, P_{10}$, then we opt for a parallel algorithm: we use an appropriate hash function on `student` at processor P_0 and distribute the tuples of `Exam` evenly among the 10 processors. For each of the processors $P_1, \ldots, P_5$, we can even avoid storing the tuples of the 100 pages, because we can simply compute the result working in the buffer. For each of the processors $P_6, \ldots, P_{10}$ we receive 100 pages and we can use a 2-pass algorithm based on sorting, producing and storing 20 sorted sublists on the fly, each one of 5 pages. After that, we can do the merging phase and produce the result.

The cost in terms of the number of page accesses is 1.000 (reading the pages of `Exam` at processor P_0) + $5 \times (100 + 100)$ (writing the 20 sublists at the processors $P_6, \ldots, P_{10}$ and then reading them in the merging phase) = 2.000. The elapsed time is 1.000 + 200 = 1.200.

Actually, we can even do better, if we observe that for each tuple of `Exam` we only need to send the value of `student` and the value of `mark`. If we do that, the cost in terms of the number of page accesses is 1.000 (reading the pages of `Exam` at processor P_0) + $5 \times (67 + 67)$ (writing the 20 sublists at the processors $P_6, \ldots, P_{10}$ and then reading them in the merging phase) = 1.670. The elapsed time is 1.000 + 134 = 1.134.

Problem 3

Consider the relations `R(A,B)` and `S(C,D,E)`, where (*i*) both relations are stored in a heap with 9.000 and 30.000 pages, respectively, (*ii*) the values appearing in the attribute `A` are the integers between 1 and 100 and such values are uniformly distributed over the tuples of `R`, (*iii*) `C` is the primary key of `S`. Knowing that the buffer has 200 frames, illustrate the algorithm you would use to answer the query

```
select A,E
from R join S on R.B = S.C
where A > 50
order by E
```

and tell which is its cost in terms of number of page accesses.

Solution 3

The query requires a join on `B = C`, followed by a sorting on `E`. It follows that we cannot combine the two operations, implying that we first have to compute the join and store the result and then sort the result itself. Note that the tuples of `R` to consider are 50% of 9.000 and this means that when reading `R` we discard 4.500 tuples. Since `C` is a key of `S`, for each tuple of `R` there is at most one tuple of `S` joining it. It follows that (*i*) the size of the result is no more than 4.500 pages, and (*ii*) we do not have the problem of large joining fragments and we can therefore use a multi-pass join algorithm based on sorting, where 2 passes suffice (because 200×199 is greater than 39.000).

It follows that the cost of the join is 9.000 (for reading `R`) + 30.000 (for reading `S`) + 4.500 (writing the sorted sublists of `R`) + 30.000 (writing the sorted sublists of `S`) + 4.500 + 30.000 (for reading the sorted sublists of both relations in order to compute the join during the merging phase) + 9.000 (for writing the result of the join) = 117.000. The cost of the sorting of the result of the join through the 2-pass algorithm is $3 \times 4.500 = 13.500$. So, the total cost is 130.500.

Problem 4

Consider the relation `PRODUCT(code,category,supplier,cost)` that stores information about products sold by an e-commerce company, with the code, the category, the supplier and the cost of each product. The relation has 3.000.000 tuples, stored in 300.000 pages, and has 10.000 different values in the attribute `cost`, uniformly distributed in the tuples of the relation. We assume that all fields and pointers have the same length, independently of the attribute and that there is a sparse, clustering B^+-tree index on `PRODUCT` with search key `cost`, using alternative 2. Consider the query asking for the code and the category of the products whose cost is in a given range constituted by 10 values, and tell how many page accesses we need for computing the answer to the query.

Solution 4

Each page contains 3.000.000/300.000 = 10 records of `PRODUCT`, which means 40 values. The index is sparse, with one data entry for each page of `PRODUCT`. Therefore, 300.000 data entries are to be stored in the leaves. Since each leaf contains 19 data entries plus one pointer to the next leaf, the number of leaves for storing

the data entries is 300.000 / 19 = 15.790. But since the pages are occupied at 66%, we need 23.685 leaves. Each intermediate node of the tree contains at most 20 index entries, and therefore, we can assume that the fan-out of the tree is (20+10)/2=15. Once we have reached the first leaf, we need another access to reach the data file. We have 3.000.000/10.000 = 300 records with the same value of cost and 10 values we have for cost. Therefore, we have to access 3.000 records of `PRODUCT`. Since each page contains 10 records, this means that we need 300 pages besides the first. In total, we have: $log_{15}23.685 + 1 + 300 = 4 + 1 + 300 = 305$ page accesses.