

Exercises on relational operators

Data Management





A.Y. 2017/18

Maurizio Lenzerini

Exercise 1

- Relation R is stored in 8.000 pages, relation S is stored in 24.000 pages, and our DBMS has 500 free buffer frames. We want to compute the bag difference $R -_B S$. Consider the following two questions:
 - A. Among the one-pass, the two-pass and the three-pass algorithm based on sorting, which one would you choose, and why?
 - B. Describe in detail the algorithm you have chosen, and tell how many page accesses it requires for computing $R -_B S$.
- Answer the above two questions in two scenarios: (A) R and S are not sorted; (B) R is not sorted, but S is sorted on all attributes.

Solution of exercise 1

-  A. R and S are not sorted. Obviously, we cannot use the one-pass algorithm based on sorting, because $\min(B(R), B(S)) \geq 500 - 1$. We can use the two-pass algorithm because $500 \times (500-3) = 248.500$, and $B(R)+B(S) = 32.000 \leq 248.500$. The two-pass algorithm based on sorting creates at most $499 \times 8.000 / 32.000 = 124$ sorted sublists for R and at most $499-124 = 275$ sorted sublists for S using 500 buffer frames. Actually, the algorithm creates $8.000/500 = 16$ sorted sublists for R, and $24.000/500 = 48$ sorted sublists for S. When we have the sorted sublists for R and the sorted sublists for S, we perform the second pass by computing the bag difference by reading each sublists one page at a time in the corresponding buffer frame, and using one output frame for the result. We do not have problems for this pass because $16 + 48 \leq 500$. The cost is $3 \times (24.000 + 8.000) = 96.000$ page accesses. 
-  B. R is not sorted, but S is sorted on all attributes. Still, we cannot use the one pass algorithm. We can use a modified version of the two-pass algorithm, where we do not produce the sorted sublists for S; rather, we create the sorted sublists for R using 500 buffer frames. We will have $8.000/500=16$ sorted sublists for R, each one sorted on all attributes (the same criterium used to sort S). In the second pass we compute the bag difference by using the appropriate 16 buffer frames for reading the sublists of R one page at a time, 1 buffer frame for reading the whole S (already sorted) one page at a time, and one output frame for the result. The cost is $B(R)+3 \times B(S) = 24.000 + 3 \times 8.000 = 48.000$ page accesses. 






Exercise 2

A relation R with attributes A,B,C is stored in 8.000 pages (with 100 tuples per page), a relation S with attributes D, E is stored in 5.000 pages (with 100 tuples per page), and our DBMS has 122 free buffer frames. We want to compute the join of R and S on the condition $C = D$, and we know that the 200 different values stored in the attribute C are uniformly distributed in the tuples of R, and the 160 different values in the attribute D are uniformly distributed in the tuples of S.

1. Can we use the block nested-loop algorithm?
2. Can we use a two-pass algorithm based on sorting?

For each of the two cases (1 and 2), (i) if the answer is no, then explain the answer; (ii) if the answer is yes, then describe in detail how the algorithm (or, for case 2, the variant chosen) works, and tell which is the cost of the algorithm for computing the above join in terms of number of page accesses.

Solution of exercise 2

-  • We can always use the block nested-loop algorithm. The cost is $5.000 + (5.000/(122 - 2) + 1) \times 8.000 = 5.000 + (41+1) \times 8.000 = 341.000$.

-  • We can use the two pass join algorithm, because $\text{ceil}(8.000/122) + \text{ceil}(5.000/122) = 107$ is less than or equal to 121. More precisely, we can use the sort-merge join algorithm, where in the first phase we produce $8.000/122 = 66$ sorted sublists for R, and $5.000/122 = 41$ sorted sublists for S. In the second phase of the algorithm we read the various sublists and we compute the join, using 107 buffer frames for the sorted sublists of R and one frame for the output. An important point is to show that the tuples of the two relations that join are such that the size of the space they require will never exceed the size of the buffer. Indeed, the number of tuples of R having the same value of C is $8.000 \times 100/200 = 4.000$ and therefore the number of pages required to store them is $4.000/100 = 40$ (which is less than 66), and the number of tuples of S having the same value of D is $5.000 \times 100/160 = 3.125$ and therefore the number of pages required to store them is $3.125/100 = 32$ (which is less than 41). The cost of the algorithm in terms of number of page accesses is $3 \times (8.000 + 5.000) = 39.000$.

-  • If we had used the simple sort-based join algorithm, then the cost would have been $5 \times (8.000 + 5.000) = 75.000$.

Exercise 3



Consider the relation TRAVEL(code, person, nation, cost) that stores information about travels of people, with the code of the travel, the person who traveled, the nation visited, and the cost of the travel. 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 relation. We assume that all fields and pointers have the same length, independently of the attribute. There is a sparse, clustering B+-tree index on TRAVEL with search key cost, using alternative 2, with one data entry for each page of TRAVEL. Consider the query that asks for code and nation of the travels 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 of exercise 3

- Each page contains $3.000.000/300.000 = 10$ records of TRAVEL, which means 40 values. How many leaves in the index?
- The index is sparse, with one data entry for each page of TRAVEL. Therefore, we need to store 300.000 data entries in the leaves. Since each leaf has room for 20 data, the number of leaves required for storing the data entries is $300.000 / 20 = 15.000$. But since the pages are occupied at 66%, we need 22.500 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, and other accesses to reach all tuples of the result. Note that in the average we have $3.000.000/10.000 = 300$ tuples with the same value of cost, and we have to consider 10 values for cost. Therefore, we have to access 3000 records of TRAVEL. Since each page contains 10 records, we need to access 300 pages of TRAVEL besides the first. In total, we have: $\log_{15} 22.500 + 1 + 300 = 4 + 1 + 300 = 305$.







Exercise 4

Relation $R(X,Y)$ occupies 20.000 pages, with 20 tuples per page, and relation $S(Y,Z)$ occupies 150.000 pages, has Y as primary key, and has a hash-based index with Y as search key. We have to compute the natural join between R and S , having 100 buffer frames available.

-  1. Describe the one-pass join algorithm, tell whether we can use this algorithm in our case, and, if yes, tell what is its cost in terms of page accesses.
2. Describe the two-pass merge-sort algorithm, tell whether we can use this algorithm in our case, and, if yes, tell what is its cost in terms of page accesses.
3. Describe the three-pass merge-sort algorithm, tell whether we can use this algorithm in our case, and, if yes, tell what is its cost in terms of page accesses.
4. Describe the block-nested loop join algorithm, tell whether we can use this algorithm in our case, and tell what is its cost in terms of page accesses.
-  5. Describe the index-based join algorithm, tell whether we can use this algorithm in our case, and tell what is its cost in terms of page accesses.

Solution of exercise 4

Relation $R(X,Y)$ occupies 20.000 pages, with 20 tuples per page, and relation $S(Y,Z)$ occupies 150.000 pages, has Y as key, and has a hash-based index with Y as search key. We have to compute the natural join between R and S , having 100 buffer frames available.

-  1. We cannot use the one-pass algorithm, because both 20.000 and 150.000 are greater than $100-2$, i.e. $20.000 > 98$ and $150.000 > 98$.
2. We cannot use the two-pass algorithm based on sorting, because $150.000/100 + 20.000/100$ is greater than $100-1$, i.e., $1.500 + 200 > 99$, i.e., $1.700 > 99$.
-  3. We can use the three-pass algorithm based on sorting, because $150.000/100 + 20.000/100$ is not greater than $(100-1)^2$, i.e., $1.700 \leq 9.801$. The cost is $5 \times (20.000 + 150.000) = 850.000$. Note that we do not have the problem of too many joining tuples in the buffer, because Y is a key for S .
-  4. We can always use the block-nested loop join algorithm, and the cost is $20.000 + 150.000 \times (20.000/99 + 1) = 20.000 + 150.000 \times (202 + 1) = 20.000 + 30.450.000 = 30.470.000$.
-  5. We can use the index-based join algorithm. Taking into account that R has 400.000 tuples, and counting 1 page access for searching for the tuple of S with a given value of Y using the hash-based index, the cost is $20.000 + 400.000 = 420.000$.

Exercise 5

Consider the relation CAR(code,owner,type,year), storing information about cars, each one with its type, its owner, and the year of its construction. CAR has 500.000 tuples stored in a heap file, where each page contains 50 tuples. Consider the aggregate query Q that, for each owner o, computes the number of cars owned by o, and assume that we have a good hash function on owner that distributes the tuples of CAR uniformly, and that we have 101 free buffer frames.

1. Which algorithm would you use for computing Q?
2. Describe in detail the algorithm chosen.
3. Tell which is the cost of executing the algorithm in terms of number of page accesses.

Solution of exercise 5

The relation is stored in $500.000/50 = 10.000$ pages. The availability of a good hash function on owner that distributes the tuples of CAR uniformly suggests the two-pass algorithm based on hashing. Indeed, we can execute such algorithm if the number of pages of the relation is less than or equal to $M \times (M - 1)$, where M is the number of free buffer frames.

In our case $M \times (M - 1) = 101 \times 100 = 10.100$, and since $10.000 \leq 10.100$, we can indeed use the two-pass algorithm based on hashing. Such algorithm uses the first pass to distribute, using the hash function on owner, the tuples of the relation in $M - 1$ buckets stored in secondary storage, in such way that tuples with the same value of owner are in the same bucket. In the second pass, we treat each bucket in isolation. For each bucket we read its pages and we store in the buffer one tuple for each value of owner, accumulating the result (in this case, the count) while reading the pages of the bucket. After having read all pages of the bucket, we write the content of the buffer in the result. The cost is obviously $3 \times 10.000 = 30.000$ page accesses (as usual, we ignore the cost of writing the final result).

Exercise 6

Assume that relation $R(A,B)$ has 10.000 tuples, relation $Q(C,D,E,F)$ has 400.000 tuples, attribute D has 2.000 values uniformly distributed on the tuples of Q , each page of our system contains 400 Bytes, every attribute value or pointer requires 20 Bytes, and we have 252 free buffer frames. Consider the query

select A

from R

where not exists (select $*$ from Q where $Q.D = R.B$)

and tell which is the algorithm you would use and the corresponding cost (in terms of number of page accesses) for executing such query for each of the following methods for representing Q :

- (1) heap file;
- (2) sorted file with sorting key D ;
- (3) heap file with an unclustering, dense sorted index with duplicates with search key D (strongly dense index);
- (4) sorted file with a clustering, sparse sorted index with search key D ;
- (5) sorted file with a clustering, dense sorted index without duplicates with search key D ;
- (6) sorted file with a clustering, sparse tree-based index with search key D .

Solution of exercise 6

Since R has 10.000 tuples and two attributes, each of 20 Bytes, and the size of each page is 400 Bytes, the number of pages of R is $10.000 \times 2 \times 20 / 400 = 1.000$. Since Q has 400.000 tuples and four attributes, each of 20 Bytes, the number of pages of Q is $400.000 \times 4 \times 20 / 400 = 80.000$.

- (1) If Q is represented as a heap file, then the query can be answered in principle by a multipass algorithm. Note that we need three passes, because $10.000 + 80.000 > 63.252$ (where $63.252 = 252 \times 251$), while $10.000 + 80.000 < 15.876.252$ (where $252 \times 251 \times 251 = 15.876.252$). The cost, however is $5 \times 90.000 = 450.000$ page accesses. It is then reasonable to see what is the cost of the block nested-loop algorithm. Using such algorithm, we load relation R in blocks, each of 250 pages, and for each block b we scan relation Q to find the tuples in b that satisfy the where condition (we use one buffer frame among the 252 free frames available for reading Q, and one for producing the output). The cost is then $1.000 + (1.000 / 250) \times 80.000 = 321.000$ page accesses, which is less than the cost of the three-pass algorithm

Solution of exercise 6

- (2) If Q is represented as a sorted file, then the query can be answered by scanning the tuples of R, and for each tuple t_1 of R, using binary search for checking whether there exists a tuple t_2 in Q such that $t_1.B = t_2.D$, including t_1 in the result if such check fails. The cost is then $1.000 + 10.000 \times \log_2 80.000 = 171.000$ page accesses.
- (3) If Q is represented as a heap file with an unclustering, dense sorted index with duplicates (i.e., strongly dense, which means that we have one data entry per data record) with search key D, then the query can be answered by means of an index-based algorithm that scans the tuples of R, and for each tuple t_1 of R uses the sorted index for checking whether there exists a tuple t_2 in Q such that $t_1.B = t_2.D$, including t_1 in the result if such check fails. Since the index is unclustering, is dense, and has duplicates, it has one data entry for each tuple in Q, i.e., it has 400.000 data entries. Each data entry requires $2 \times 20 = 40$ Bytes, and therefore each page has 10 data entries, and the number of pages of the index is 40.000. It follows that the cost is $1.000 + 10.000 \times \log_2 40.000 = 161.000$ page accesses.

Solution of exercise 6

- (4) If Q has a clustering, sparse sorted index with search key D, then the query can be answered again by means of an index-based index algorithm, as before. Since the index is clustering and sparse, it has one data entry for each page of Q. Since each page has 10 data entries, as we saw before, the number of pages of the index is $80.000 / 10 = 8.000$. Note that in this case, after accessing the index, we have to follow the pointer to the data file, since the index is sparse. It follows that the cost is $1.000 + 10.000 \times (\log_2 8.000 + 1) = 141.000$ page accesses.
- (5) If Q has a clustering, dense sorted index without duplicates (i.e., we have one data entry for each value of the search key) with search key D, then the query can be answered by means of an index-based algorithm, as before. Since the index is clustering and dense, it can avoid duplicates, and therefore it has one data entry for each value in D, i.e., 2.000. Since each page has 10 data entries, as we saw before, the number of pages of the index is $2.000 / 10 = 200$. It follows that the cost is $1.000 + 10.000 \times \log_2 200 = 81.000$ page accesses.

Solution of exercise 6

- (6) If Q has a clustering, sparse tree-based index with search key D, then the query can be answered again by means of an index-based index algorithm, as before. Since the index is clustering and sparse, it has one data entry for each page of Q. Since each page has room for 10 data entries, taking into account the 67% rule, we count 6 data entries per page, and thus the number of leaves of the index is $80.000 / 6 = 1.333$. Also, since each intermediate node has room for 10 index entries, we can count 7 as fan-out. Note that in this case too, after accessing the index, we have to follow the pointer to the data file, since the index is sparse. It follows that the cost is $1.000 + 10.000 \times (\log_{10} 1.333 + 1) = 51.000$ page accesses.

Exercise 7

The SQL table R(A,B,C) has 1.400.000 tuples stored in a heap file, the SQL table Q(E,F,G,H,L) has 2.400.000 tuples stored in a heap file, and there is a B+-tree index on (E,F), where (E,F) is the key of Q.

We know that each attribute or pointer occupies 20 Bytes, the size of each page is 600 Bytes, and the buffer has 400 free frames. Consider the following SQL query (we remind the student that the minus clause in SQL computes the difference by eliminating duplicates, i.e., by taking the distinct rows of the first operand and returning the rows that do not appear in a second operand):

```
select T.A, T.B
from (select A as A1, B as A2 from R
      minus
      select E as A1, F as A2 from Q) as T
order by T.A, T.B
```

Describe in detail the algorithm you would use to compute the answer to the above query, and tell which is the cost of the algorithm in terms of number of page accesses.

Solution of exercise 7

Each page has space for $600/60 = 10$ tuples of R, and therefore R is stored in a heap with $1.400.000/10 = 140.000$ pages. Similarly, each page has space for $600/100 = 6$ tuples of Q, and therefore Q is stored in a heap with $2.400.000/6 = 400.000$ pages.

Note that the B+-tree index on (E,F), where (E,F) is the key of Q, is unclustering, and therefore dense. Since each page has space for $600/40 = 15$ data entries of such index, taking into account the 67% occupancy rule, we know that each page contains 10 data entries, implying that the B+-tree index has $2.400.000/10 = 240.000$ leaf pages.

Notice that we cannot use the one-pass algorithm, and that $400 \times 399 = 140.000$ is less than 159.600. Also the sorted projection Q on E,F is directly available in the leaves of the B+-tree index on Q with search key (E,F). It follows that in order to compute the result of the query, we can use a variant of the two-pass algorithm for bag difference based on sorting, by first producing the sorted sublists for the projection of R without duplicates, and then applying the second pass to compute the difference without duplicates using directly the leaves of the B+-tree index on Q with search key (E,F).

Solution of exercise 7

More precisely, the algorithm is as follows:

Pass 1: Read 399 pages of R in the buffer using one input buffer frame, sort the projection on A,B of the tuples contained in such pages, at the same time eliminating duplicates, and produce a sorted sublist of the projection of R. Note that in the worst case (the case where we do not have duplicates in R) the size of the projection of R on attributes A,B is $1.400.000 / (600/40) = 93.334$. Therefore pass 1 requires to access 140.000 pages for reading R, and 93.334 pages for writing the $140.000/399 = 359$ sorted sublists containing the projection of R on attributes A,B without duplicates.

Pass 2: Use the free buffer frames for loading one page at a time the 359 sorted sublists of the projection of R on attributes A,B, and one page at a time of the leaves of the B+-tree index on Q with search key (E,F). At each stage, we analyze the first (according to the sorting) tuple stored in the pages devoted to the projection of R, and we write it in the output frame only if it does not appear in the page devoted to the leaves of the B+-tree index. If, on the contrary, such a tuple appears in the page devoted to the leaves of the B+-tree index, then we ignore it. During such a process, whenever the output frame is full, we copy it in the file corresponding to the final result.

As for the cost, pass 2 requires to read $93.334 + 240.000$ pages. The total cost of the algorithm is $140.000 + 93.334 + 93.334 + 240.000 = 566.668$ page accesses, where, as usual, we have ignored the cost of writing the final result.

Exercise 8

In the two-pass algorithms based on sorting for duplicate elimination, we assumed that we know a priori the number of buffer frames available, and that such number never changes during execution of the algorithm. In reality, the number of buffer frames available may change during the execution of the algorithm.

Tell how would you change the algorithm in order to cope with this problem.

Solution of exercise 8

If the number of available buffer frames for the two-pass algorithms based on sorting for duplicate elimination changes, then this means that in the second pass we may have more sublists than the available buffer frames, and we may have sublists of different size (each of size less than or equal to M). In this case, we can always recursively choose M sublists (where M is less than the number of frames currently available) and merge them into another sublist. The recursion may stop when we arrive at $M-1$ sublists, where M is the number of frames currently available. At this point we perform the final merge for eliminating the duplicates.

The cost increases, because for the sublists to be merged before the final pass we have to read and write the corresponding pages once.

Exercise 9

In the two-pass algorithms based on hashing for duplicate elimination, we assumed that we know a priori the number of buffer frames available, and that such number never changes during execution of the algorithm. In reality, the number of buffer frames available may change after the execution of the first pass of the algorithm, in particular from bucket to bucket. Also, although the algorithm assumes that each of the $M-1$ buckets for R contains $B(R)/(M-1)$ pages, it may happen that the number of pages in one bucket is different from the number of another bucket.

Tell how would you change the algorithm in order to cope with these two factors.

Solution of exercise 9

In the original formulation of the algorithm, in the second pass we consider one bucket B_i at a time, and we perform duplicate elimination on B_i in one pass.

In reality, if B_i does not fit in the buffer (either because it is too big, or because some of the original frames are not available), we can apply duplicate elimination in two-pass for B_i , counting on the fact that the number of frames are sufficient for the two-pass version of the algorithm. Obviously, in the first pass of the two-pass duplicate elimination algorithm applied to B_i , we must use a different hash function with respect to the hash function used in the first pass of the duplicate elimination algorithm for the entire relation. Such new hash function is specific for B_i .