# Exercises on relational operators

Data Management

A.Y. 2018/19

Maurizio Lenzerini

# Exercise 2 of the slides on part 6 – evaluation of relational operators

(a) Suppose that R and S are sorted (e.g., on the primary key, or on the whole attributes). Do the one-pass algorithms for

- (set or bag) union,
- (set or bag) difference,
- (set or bag) intersection,
- cartesian product,
- natural join

change with respect the ``standard'' formulation?

(b) Pick up 2 or 3 operators that we have analyzed, and write the iterator for it, thus coming up with the corresponding algorithm for tuple-by-tuple computation.

# Solution to Exercise 2 of the slides on part 6 – evaluation of relational operators

- tuple(R)

  denotes the tuple of R currently under analysis, or NULL if the tuples of R are exausted
- move(tuple(R))

  considers the next tuple of R as the current tuple of R, if we have at least one not yet considered in the current frame, otherwise it loads the next page of R in the corresponding frame, and it considers the first tuple in such page, if we have more pages to consider. If we do not have more pages to consider, then it set tuple(R) to NULL
- output(tuple(R))

  copies the current tuple of R in the output buffer frame, and if this frame is full, then writes it in the result in secondary storage
- by tuple(S) = tuple(R) we mean that the two tuples are at the same level in the sorting order; by tuple(S) < tuple(R) we mean that tuple(S) comes before tuple(R) in the ordering, and by tuple(R) < tuple(S) we mean that tuple(R) comes before tuple(S) in the ordering.

# Solution to Exercise 2 of the slides on part 6 – evaluation of relational operators

- Set union

  load the first page of R in the frame for R and load the first page of S in the frame for S

  while true do {

  if no tuple(S) and no tuple(R) then EXIT

  if no tuple(S) then copy R into the output and EXIT

  if no tuple(R) then copy S into the output and EXIT

  if tuple(S) = tuple(R)

  then output(tuple(S)), move(tuple(S)), and move(tuple(R))

  else if tuple(S) < tuple(R)

  then output(tuple(S)), and move(tuple(S))

  else output(tuple(R)), and move(tuple(R))

  }

Space requirement:

- 2 buffer frames for reading S and R,
- 1 buffer frame for the output.

# Solution to Exercise 2 of the slides on part 6 – evaluation of relational operators

- Set difference (S – R)

```
load the first page of R in the frame for R and load the first page of S in the
frame for S
while true do {
    if no tuple(S) then EXIT
    if no tuple(R) then copy S into the output and EXIT
    if tuple(S) = tuple(R)
    then move(tuple(S)), and move(tuple(R))
    else if tuple(S) < tuple(R)
            then output(tuple(S)), move(tuple(S))
             else move(tuple(R))
}
```

Space requirement:

- 2 buffer frames for reading S and R,
- 1 buffer frame for the output.

# Solution to Exercise 2 of the slides on part 6 – evaluation of relational operators

- Bag difference (S – R): same as set difference!

Space requirement:

- 2 buffer frames for reading S and R,
- 1 buffer frame for the output.

# Solution to Exercise 2 of the slides on part 6 – evaluation of relational operators

- Set intersection

  load the first page of R in the frame for R and load the first page of S in the frame for S

  while true do {

  if no tuple(S) or no tuple(R) then EXIT

  if tuple(S) = tuple(R)

  then output(tuple(S)), move(tuple(S)), and move(tuple(R))

  else if tuple(S) < tuple(R)

  then move(tuple(S))

  else move(tuple(R))

  }

Space requirement:

- 2 buffer frames for reading S and R,
- 1 buffer frame for the output.

# Solution to Exercise 2 of the slides on part 6 – evaluation of relational operators

- Bag intersection: same as set intersection!

Space requirement:

- 2 buffer frames for reading S and R,
- 1 buffer frame for the output.

# Solution to Exercise 2 of the slides on part 6 – evaluation of relational operators

- Cartesian product

  The algorithm does not change!

# Solution to Exercise 2 of the slides on part 6 – evaluation of relational operators

- Natural join of S(X,Y) and R(Y,Z)

  Suppose that S and R are sorted on Y. We can store S sorted in the buffer, so when we read R we compute the joining tuples more efficiently.

  load S in the buffer

  while true do {

  while tuple(R) exists and tuple(S) exists and tuple(S) = tuple(R) do {

  output(join(tuple(S),tuple(R))), move(tuple(R))

  }

  if no tuple(R) then EXIT

  if tuple(S) < tuple(R)

  then move(tuple(S))

  else move(tuple(R))

  }

Space requirement:

- B(S) + 2 buffer frames, where S is the smaller relation

# Solution to Exercise 2 of the slides on part 6 – evaluation of relational operators

- Natural join of S(X,Y) and R(Y,Z)

  Suppose that S and R sorted on Y, and suppose that Y is a key for S.

```
while true do {
  if no tuple(R) or no tuple(S) then EXIT
  if tuple(S) = tuple(R)
  then output(join(tuple(S),tuple(R))), move(tuple(S))
  if tuple(S) < tuple(R)
  then move(tuple(S))
  else move(tuple(R))
}
```

Space requirement:

- 2 buffer frames for reading S and R,
- 1 buffer frame for the output.

# 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 and the two-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)) ≥ 500 - 1. We can use the two-pass algorithm because 8000/500 + 24000/500 <= 499. The two-pass algorithm based on sorting 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 <= 499. The cost is 3 × (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(S) + 3 × B(R) = 24.000 + 3 × 8.000 = 48.000 page accesses.

# Exercise 2

A relation R(A,B,C) is stored in 8.000 pages (with 100 tuples per page), a relation S(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 400 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) × 8.000 = 5.000 + (41+1) × 8.000 = 341.000.
- Since 8.000 <= $122^2$ and 5.000 <= $122^2$, and also ceil(8.000/122) + ceil(5.000/122) = 107 is less than or equal to 121, we can in principle use any of the two variants of the two-pass join algorithm. In order to choose the variant (simple-sort join, or sort-merge join algorithm), we should verify whether we can use the most efficient one i.e., the sort-merge join algorithm, i.e., we should be sure that, for no value of the joining attributes, the size of the tuples with such value that we should keep in the buffer will never exceed the size of the buffer. Now, the number of tuples of R having the same value of C is 8.000 × 100/400 = 2.000 and therefore the number of pages required to store them is 21 (since 2.000/100 = 20), and the number of tuples of S having the same value of D is 5.000 × 100/160 = 3.125 and therefore the number of pages required to store them is 3.125/100 = 32. If we use the sort-merge join algorithm, 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 whould need 66 + 21 + 41 = 128 buffer frames + 1 for the output. Since we only have 122 frames, we should choose the variant of simple-sort join, because we are sure that for every value of C, all the 21 pages of R with tuples with that value in C will fit in the buffer (since 20 < 122 – 2)
- The cost of the algorithm in terms of number of page accesses is 5 × (8.000 + 5.000) = 75.000.

# Exercise 3

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 3

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 - 1) × (M − 1), where M is the number of free buffer frames.

In our case (M - 1) × (M − 1) = 100 × 100 = 10.000, 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 × 10.000 = 30.000 page accesses (as usual, we ignore the cost of writing the final result).

# Exercise 4

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 4

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 then 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 5

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 5

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 $B_i$.