Exercises on query plans

Data Management

A.Y. 2018/19

Maurizio Lenzerini

Exercise 1

Consider the relations Tournament(name,year,city,director,prize,winner), which contains 600 pages storing information about 30.000 tennis tournaments, and Player(code,lastname,yearOfBirth,cityOfBirth), which stores information about 150.000 tennis players. Assume that every attribute of every relation and every pointer has the same size, and that there is a tree-based, primary, unclustered index on Player with search key code.

Consider the query

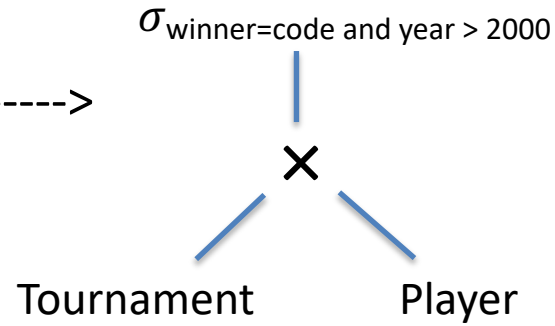
```
select *
from Tournament, Player
where winner = code and year > 2000
```

show the associated logical query plan, and, assuming that you can only use two buffer frames (besides what is needed for handling the output), tell

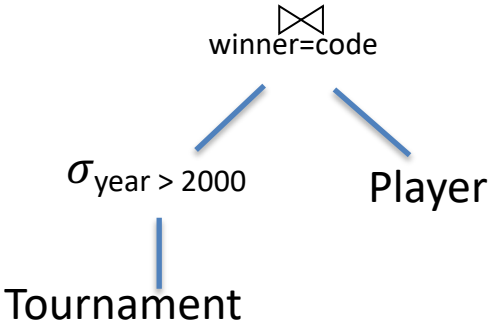
1. which physical query plan would you use for computing the answer to the above query;
2. which is the cost of computing the answer to the above query using the chosen query plan.

Solution exercise 1

Here is the logical query plan associated to the query ------->



We immediately split and push selection, and introduce the join operator ------->



Solution exercise 1

A possible, reasoning algorithm for the join is the index-based algorithm: we scan the pages of Tournament, and for each tuple t of Tournament that we find in such pages, we filter out those with year <= 2000, and we use the index to find the tuple p of Player such that t.winner = p.code, and to build the corresponding tuple in the output. Note that the algorithm uses only two buffer frames, besides what is needed for handling the output.

To come up with the cost of computing the answer to the above query using the chosen algorithm, we have to compute the number of page accesses we need in order to find a tuple of Player given a value of the attribute Code, using the tree index.

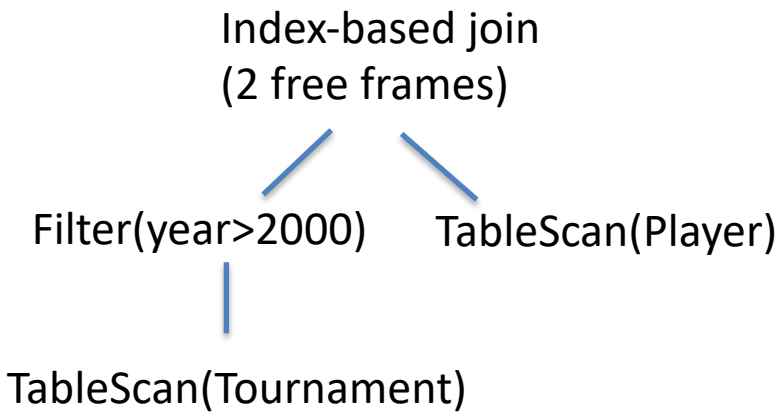
In order to do so, we need to compute the number of leaves in the tree. We obviously assume that the index uses alternative 2. The leaves of the tree must store 150.000 data entries of the form <v, p>, where v is a value of the search key, and p is a pointer. How many pages do we need to store 150.000 data entries of such form? Since 600 pages are sufficient to store 30.000 tuples of Tournament (each one with 4 fields), we infer that 50 tuples of Tournament fit in one page, and therefore 150 data entries (each one with 2 fields) of the tree index fit in one page. At this point we know that we need 150.000 / 150 = 1000 pages for the leaves of the tree, and taking into account the 67% rule occupancy, we conclude that the leaf pages of the tree index are 1500. Also, we know that 150 is the number of index entries (each one with 2 fields) that fit in one page, and therefore we can assume that the fan-out of the tree index is 112 (the average value between 75 and 150). This means that the number of page accesses we need to reach the correct data entry of the index, given a value of the attribute Code, is $\log_{112} 1500 = 3$, and the number of page accesses we need to find the corresponding tuple of Player is $\log_{112} 1500 + 1 = 4$ (we have + 1 because we have to reach the data record in the data file).

We are now ready to conclude that the cost of the algorithm, measured in terms of the number of pages accessed during its execution (ignoring as usual the cost for handling the output), is

$600 + 30.000 \times 4 = 120.600$.

Solution exercise 1

Here is the physical query plan ------->



Exercise 2

Let R(a,b) and S(b,c) two relations, with

T(R) = 5000 T(S) = 2000

V(R,a) = 50 V(S,b) = 200

V(R,b) = 100 V(S,c) = 100

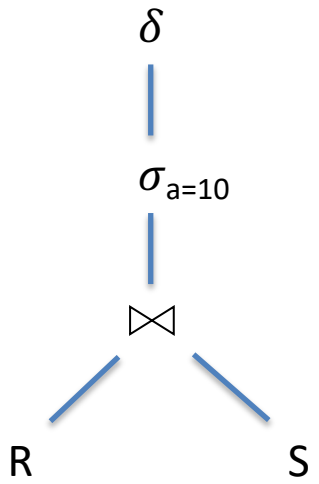
By referring to the query:

```
select distinct *
from R join S
where R.a = 10
```

- produce the associated logical query plan and analyze possible alternative logical query plans,
- show for each logical query plan an estimation of the number of tuples in all the nodes except for the root.

Solution exercise 2

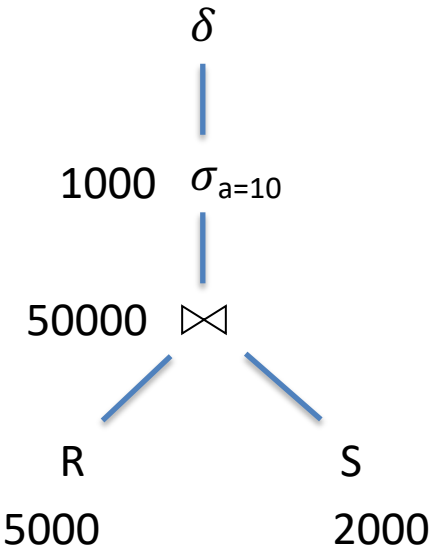
Here is the logical query plan associated to the query ------->



To estimate the number of tuples of the join J we multiply T(R)/V(R,a) by T(S), and then divide by max{V(R,b),V(S,b)} →

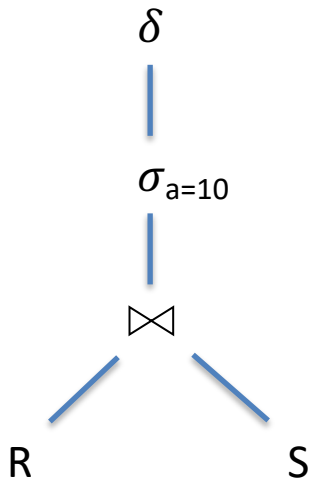
5000 × 2000 / 200 = 50000

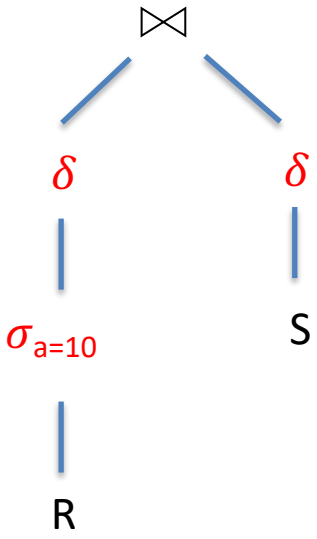
Remember that we assume that V(J,) = V(R,a)=50. To estimate the size of $\sigma_{a=10}(J)$ we divide T(J) by V(J,a) → 50000/50 = 1000



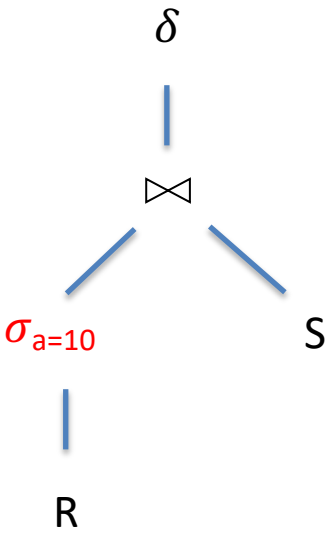
Solution exercise 2

If we push selection, we are left with the following alternatives: (a), (b), (c) and (d)

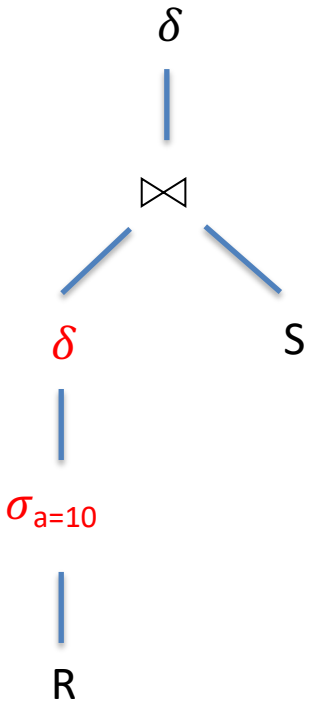




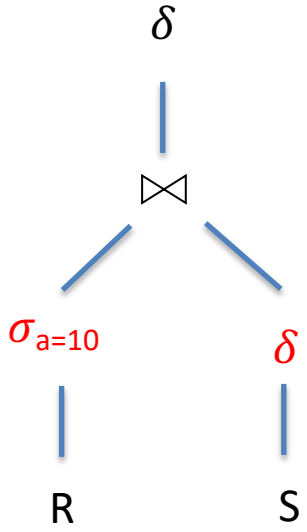
(a)



(b)



(c)



(d)

Solution exercise 2

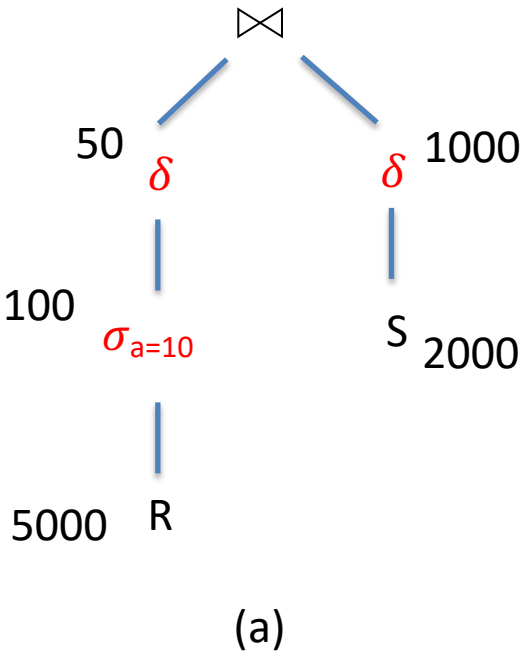
Size of $\sigma_{a=10}(R)$: we divide T(R) by V(R,a) → 5000/50 = 100

Size of join of tree (b): We multiple T(R)/V(R,a) by T(S), and divide by max{V(R,b),V(S,b)} → 100 × 2000 / 200 = 1000

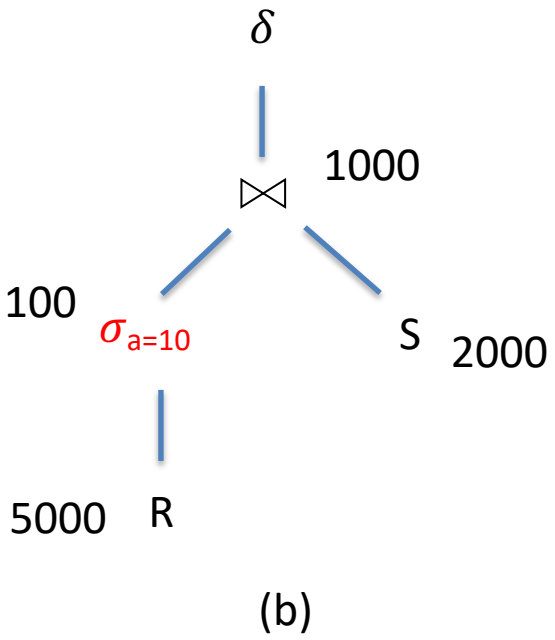
The first duplicate elimination in tree (a) → half of the tuples in $\sigma_{a=10}(R)$ = 50;

The second duplicate elimination in tree (a) → half of the tuple of S = 1000.

We do not consider the number of tuples in the root.

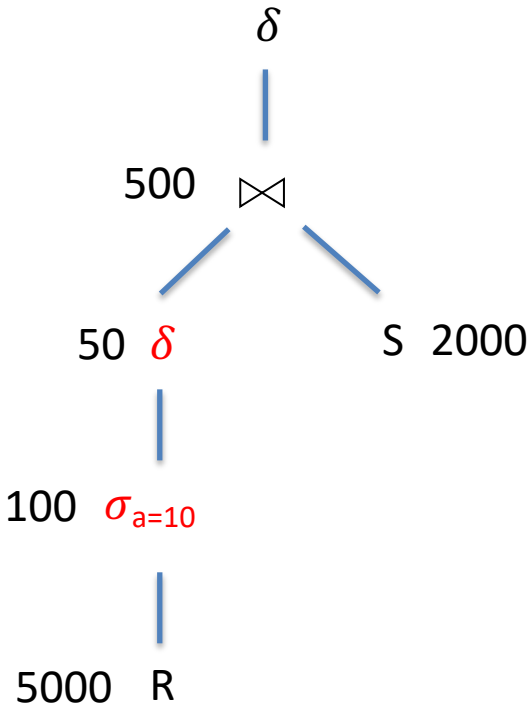


(a)

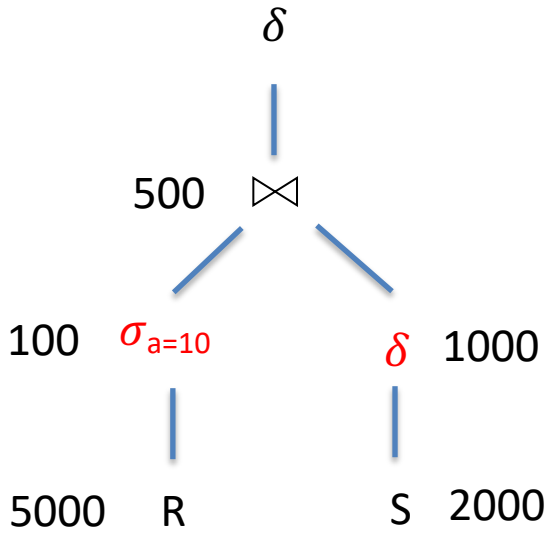


(b)

Solution exercise 2



(c)



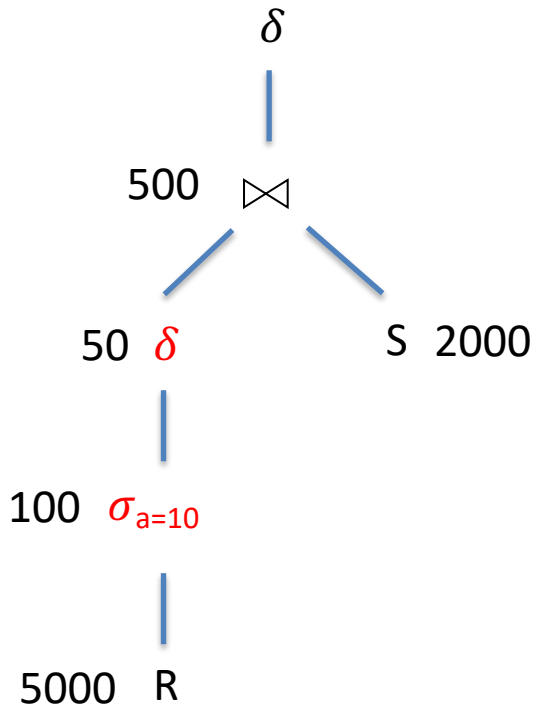
(d)

Exercise 3

Let R(a,b) and S(b,c) two relations, with

T(R) = 5000 T(S) = 2000 V(R,a) = 50 V(S,b) = 200 V(R,b) = 100 V(S,c) = 100

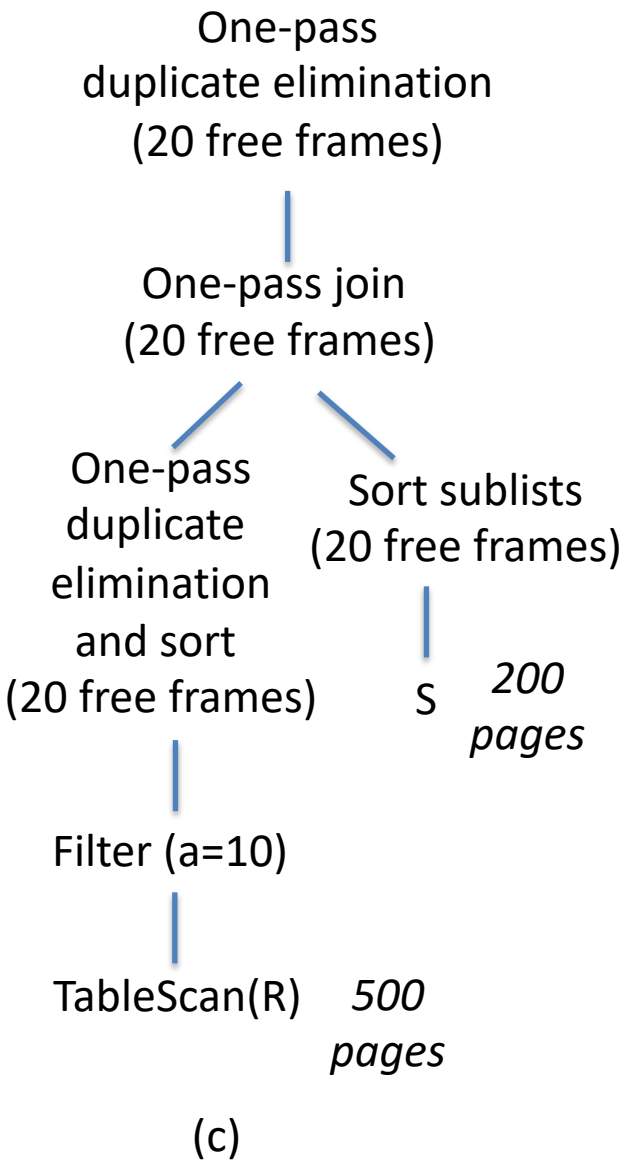
and such that the values of b are uniformly distributed on R and S. Consider the logical query plan shown below, and tell which is the physical query plan associated to it, assuming that, for both R and S, each page contains 10 tuples, and we have 20 buffer frames available. Also, compute the cost of the physical query plan.



(c)

Solution exercise 3

The corresponding physical query plan is:



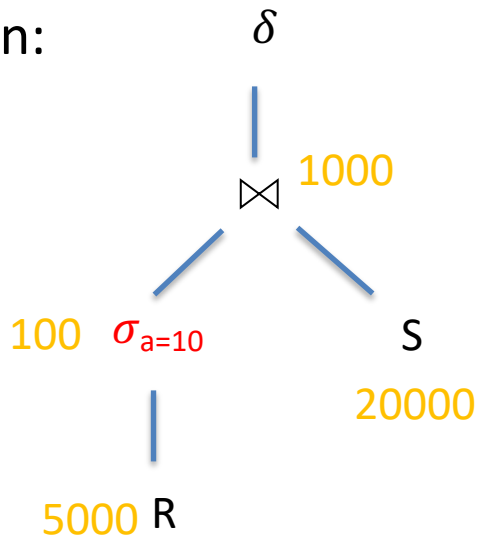
(c)

We read R (500 pages), we filter on a=10 and then we store in the buffer the tuples without duplicates (5 pages), sorted on b. We read S (200 pages) and we produce 200/20 = 10 sorted (on b) sublists, each of 20 pages. We then perform the last «merge» step of the join in one pass, by reading again the 10 sublists and eliminating duplicates while writing on the output frame.

Note that, since T(S) = 2000, and V(S,B)= 200, an average of 2000/200= 10 tuples of S will have the same value of attribute b, and therefore we do not have the problem of too many joining tuples in the buffer. The cost is 500 + 200 + 200 + 200 = 1.100 page accesses.

Exercise 4

Consider the logical query plan:



Also, let the statistics for R(a,b) and S(b,c)

be as follows:

T(R) = 5000 T(S) = 20000

V(R,a) = 50 V(S,b) = 200

V(R,b) = 100 V(S,c) = 100

We know 10 tuples of both R and S fit in

one page, and the buffer has 12 free frames.

Which physical query plan correspond to the above logical query plan?
Which is the cost of executing such plan in terms of number of page accesses?

Solution exercise 4

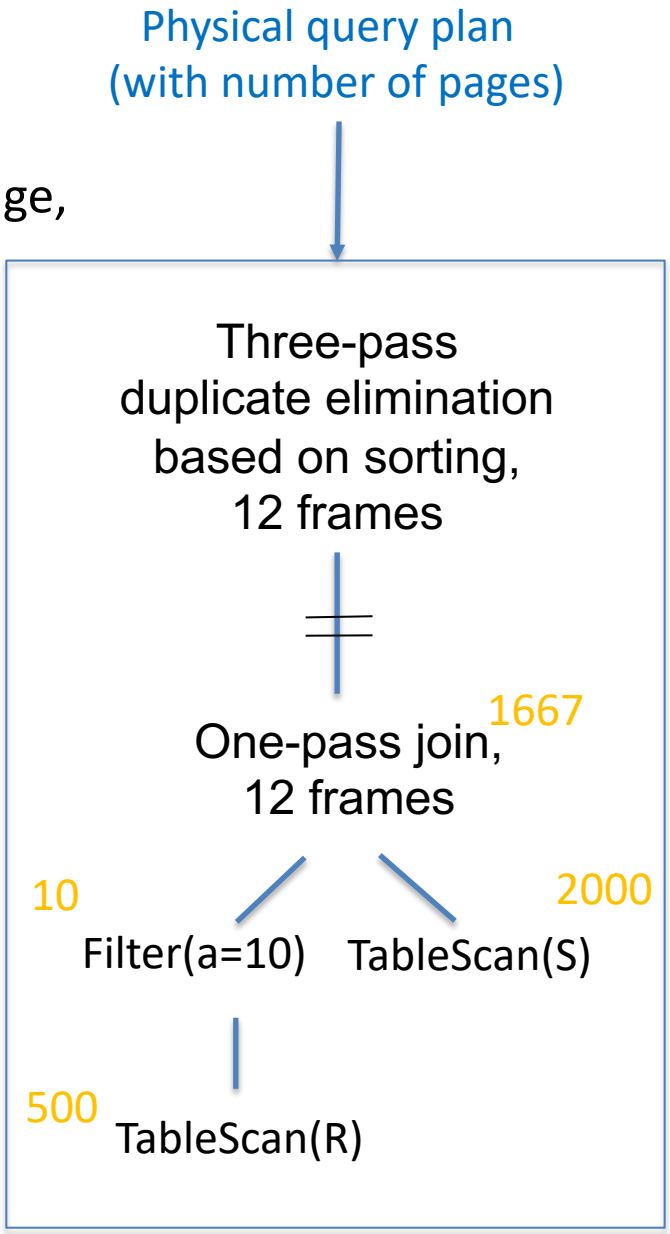
We know 10 tuples of both R and S fit in one page, and the buffer has 12 free frames. So, the pages after the selection are 10, and we can compute the join in one pass. If 10 tuples of 2 values fit in one page, 6 tuples of 3 values fit in one page. It follows that the 100 × 20000 /200 tuples of the result of the join must be materialized in 10000 / 6 = 1667 pages. Unfortunately, we cannot do duplicate elimination in one or two passes. We can do it three passes (12×12×12 = 1728 > 1667).

Cost:

500 + 2000 + 10000/6 + 5×10000/6 =

500 + 2000 + 1667 + 8335 = 12.502

TableScan(R) TableScan(S) writing of join three-pass duplicate elimination



Exercise 5

Consider the query:

```
select T
from MovieStar, StarsIn
where B = 1960 and N = SN
```

MovieStar(N, A, G, B),

StarsIn(T, SN)

T(MovieStar) = 250.000 T(StarsIn) = 2.500.000

V(MovieStar,N) = 250.000 V(MovieStar,B) = 10.000

V(StarsIn,SN) = 250.000

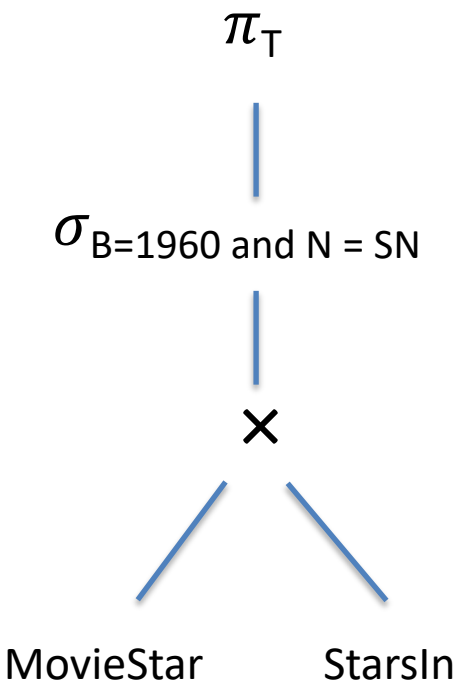
- Show the logical query plan corresponding to the query
- Choose the “best” logical query plan
- Choose the physical query plan, assuming that 100 tuples of both R and S fit in one page, and the buffer has 12 free frames.
- Compute the cost of the plan

Solution exercise 5

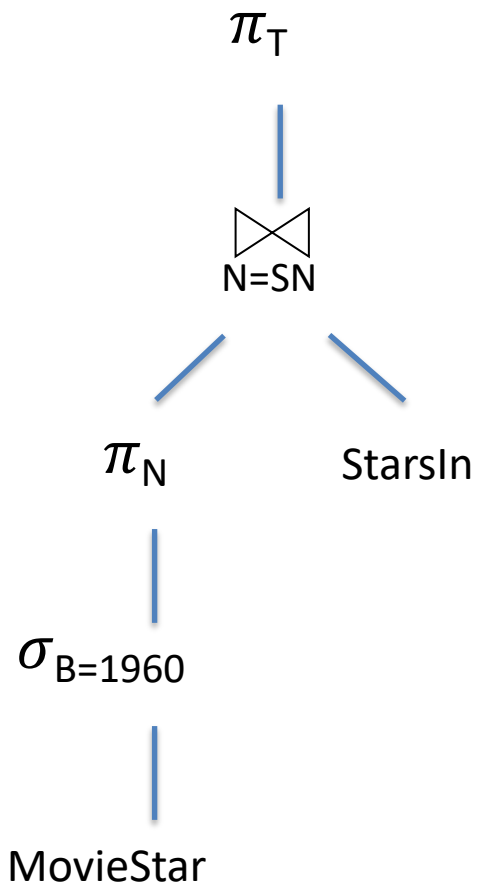
select T

from MovieStar, StarsIn

where B = 1960 and N = SN



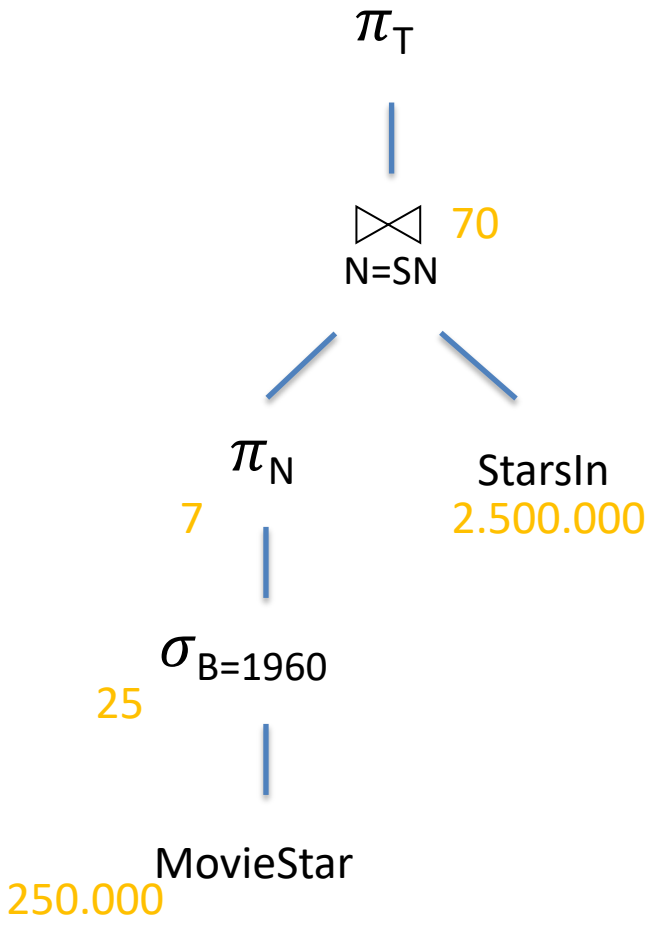
Logical query plan associated to the query



«Best» logical query plan

Solution exercise 5

Consider the logical query plan where we show the number of tuples for each node:



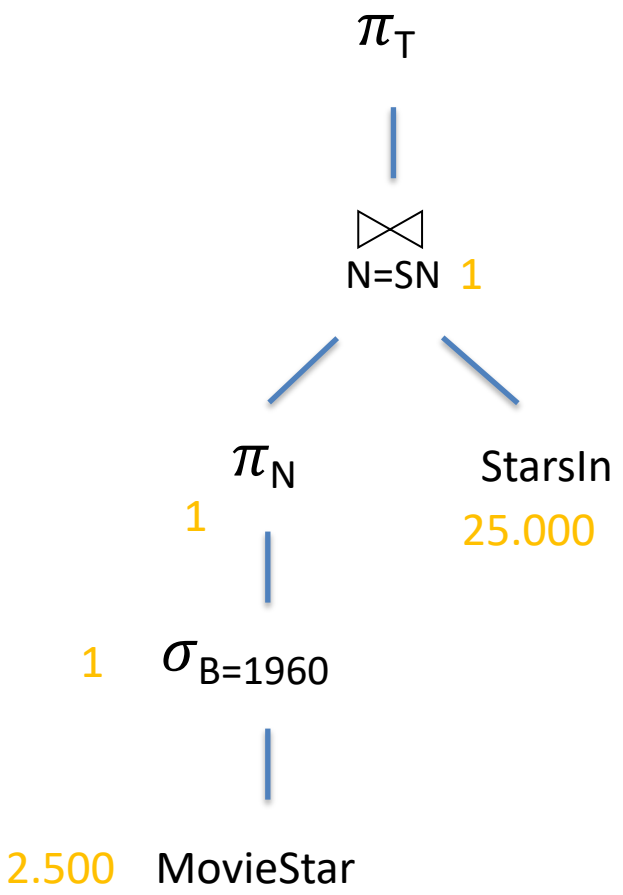

```
select T
from MovieStar, StarsIn
where B = 1960 and N = SN
```

We know that 100 tuples of both R and S fit in one page, and the buffer has 12 free frames.

Solution exercise 5

Consider the logical query plan
where now we indicate the number
of pages associated to the various nodes:

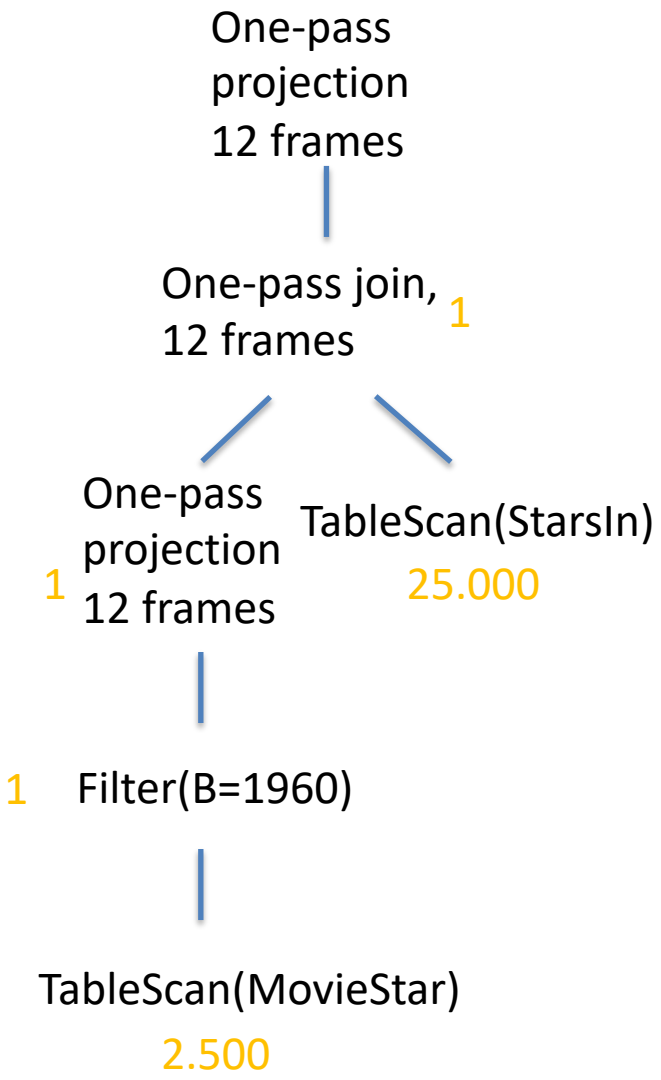
select T
from MovieStar, StarsIn
where B = 1960 and N = SN



Solution exercise 5

Here is the physical query plan:

```
select T
from MovieStar, StarsIn
where B = 1960 and N = SN
```



Cost

2.500 + 25.000 = 27.500

Exercise 6

Consider the relations

Match(Team1,Team2,Date,Result,Referee,Attendees),

Rivalry(FirstTeam,SecondTeam,Degree,Official,Timestamp)

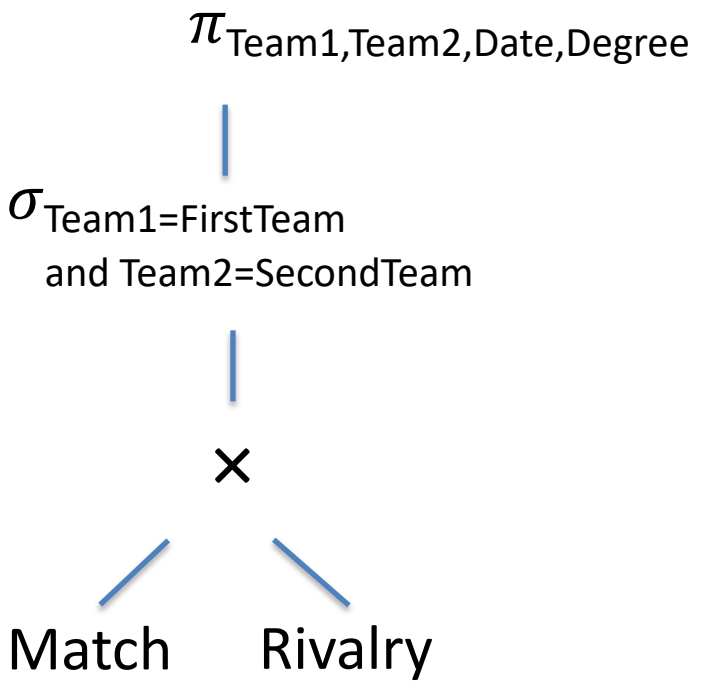
where Match occupies 2.000 pages, and Rivalry occupies 16.000 pages. We know that we have an unclustering tree index on Match with search key (Team1,Team2,Date), and we have 100 free frames in the buffer. Consider the query:

```
select Team1, Team2, Date, Degree
from Match, Rivalry
where Team1 = FirstTeam
        and Team2 = SecondTeam
```

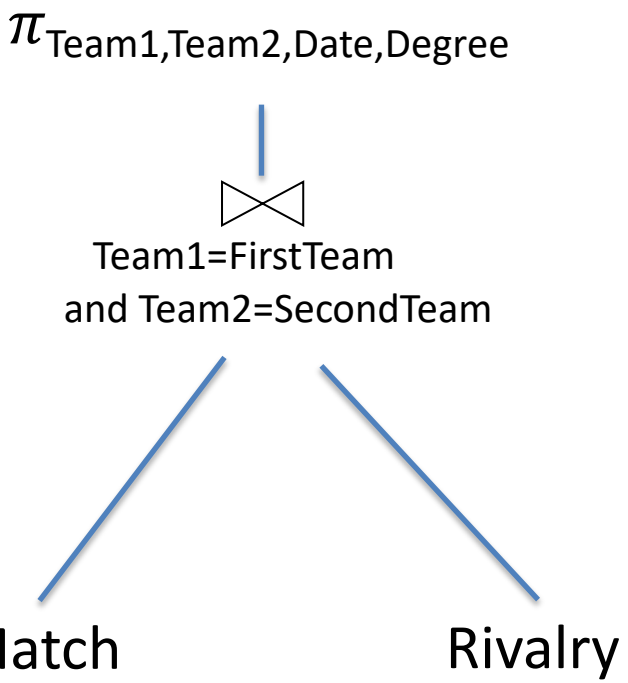
Tell which physical plan would you associate to the query, and which is its cost

Solution exercise 6

Here is the logical query plan associated to the query code:



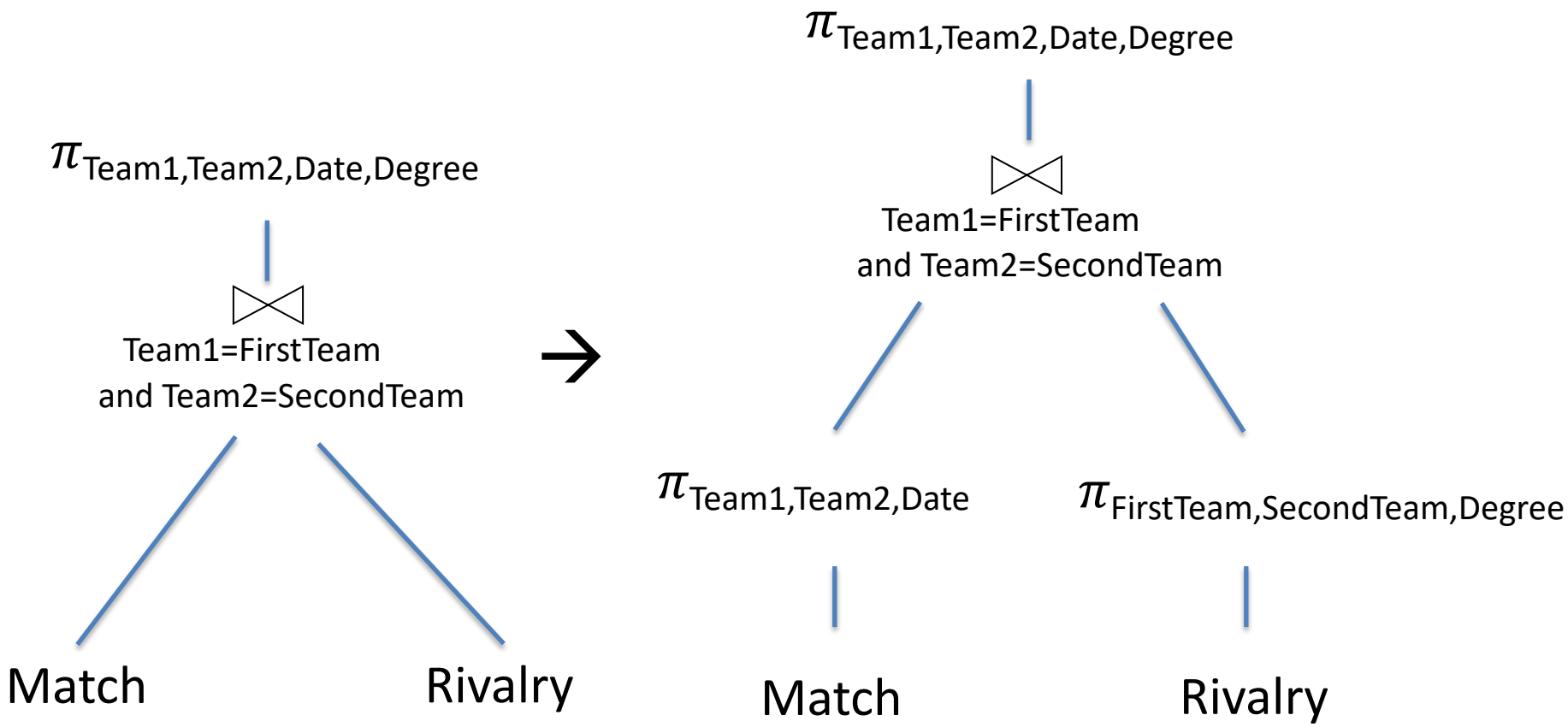
Transformed immediately as follows:



Notice that the size of every data entry of the tree index on Match is a half of a tuple, and therefore the tree index occupies 2.000 pages (taking into account the 67% rule). Since Rivalry occupies 16.000 pages, we cannot use the one-pass algorithm for the join, even if we opt for index-only scan for Match. Nor we can use the two-pass join based on sorting, because 2.000 + 16.000 > 99 × 100 (i.e., 9.900).

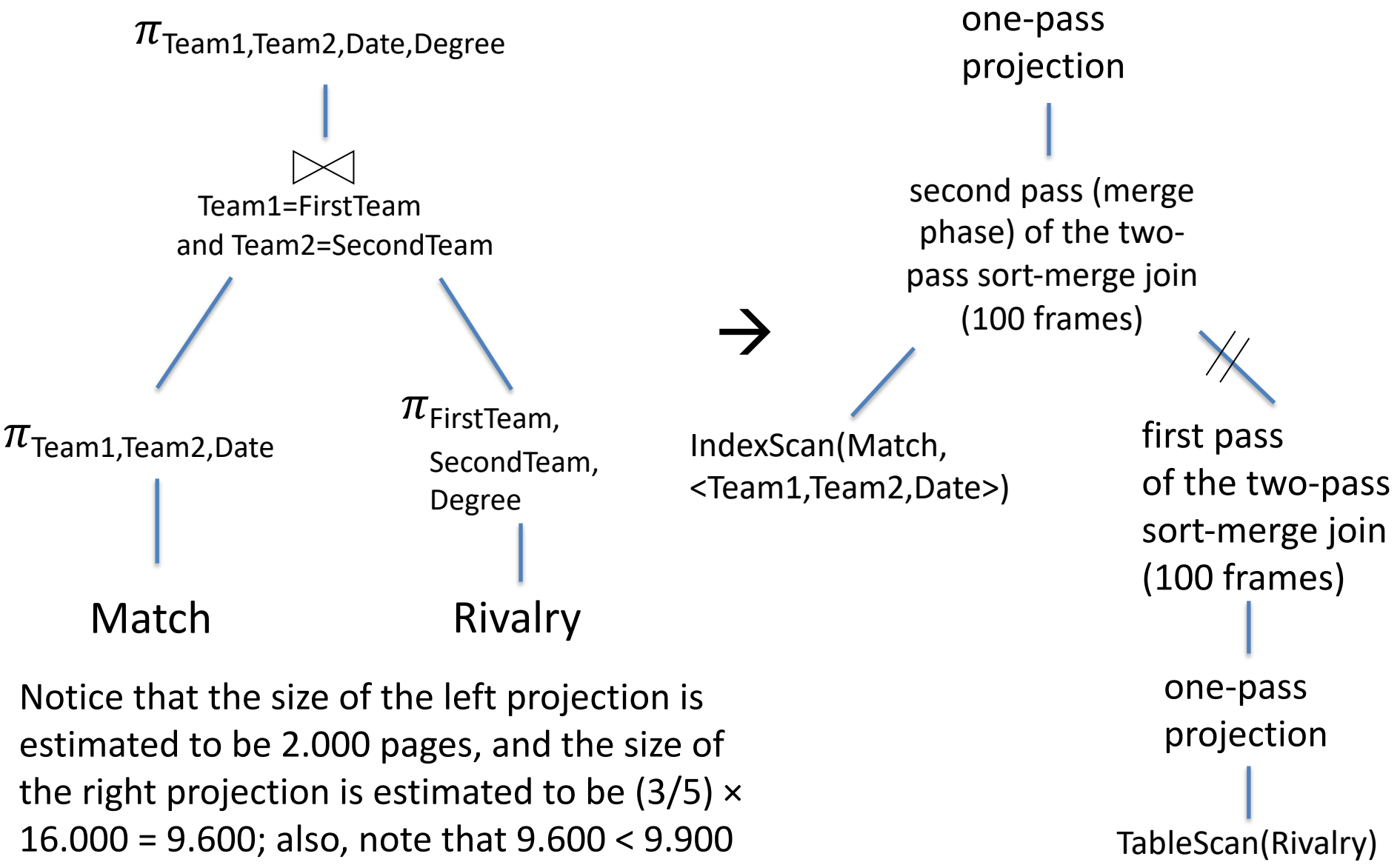
Solution exercise 6

Let us try to push projection:



Solution exercise 6

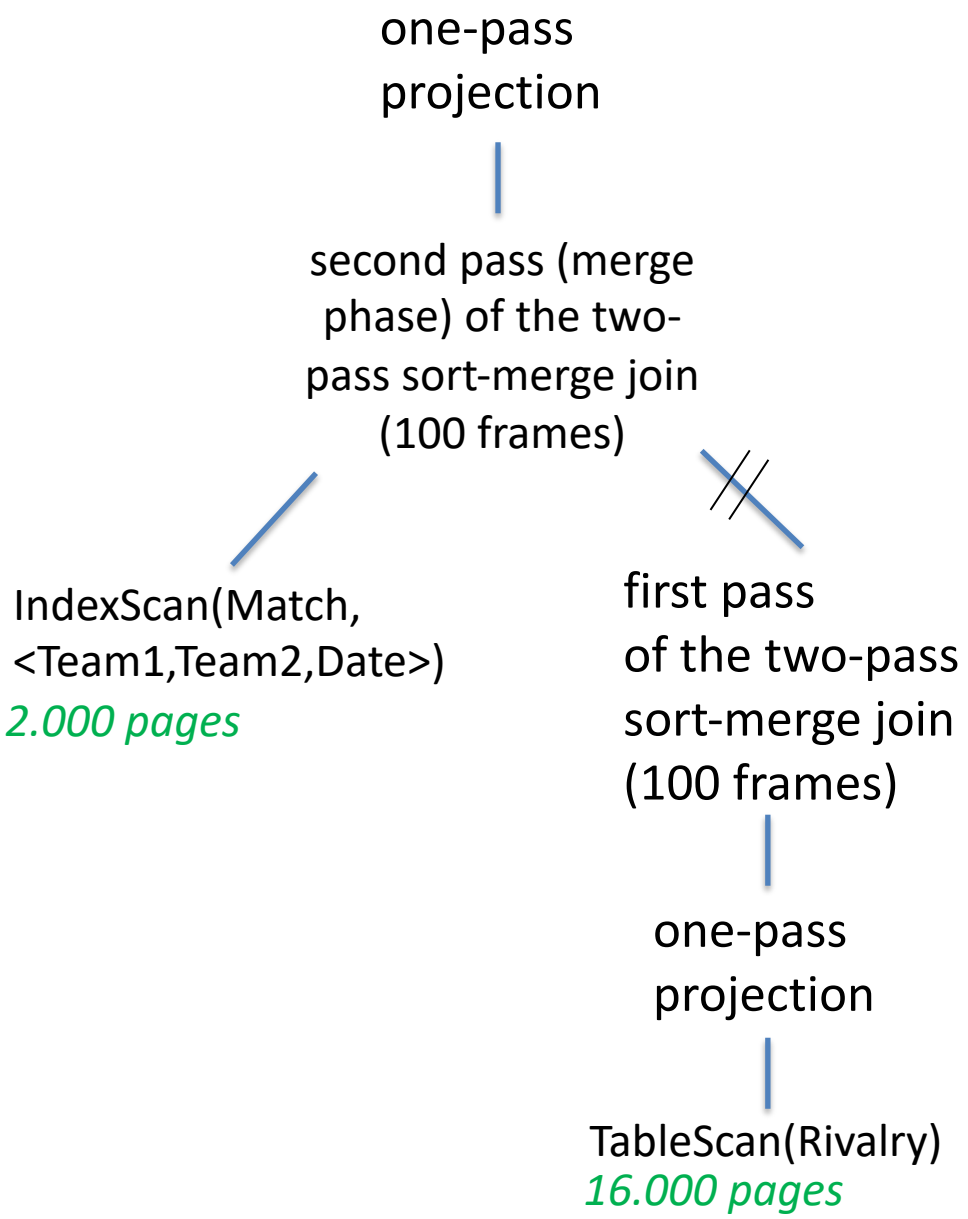
Let us now concentrate on the physical query plan.



Notice that the size of the left projection is estimated to be 2.000 pages, and the size of the right projection is estimated to be (3/5) × 16.000 = 9.600; also, note that 9.600 < 9.900

Solution exercise 6

- *reads 16.000 pages, projects the tuples while reading, and writes the 9.600/100 = 96 sorted sublists of 100 pages each*
- *reads 97 sorted files (2.000 + 9.600 pages) using 97 buffer frames, merge them, using one buffer frames for computing the final projection*



The total cost is:

16.000 + 9.600 + 9.600 + 2.000 =

37.200 page accesses

Exercise 7

Assume that relation Customer(firstName,lastName,yearOfBirth,salary) (where lastName is a key), has 650.000 tuples, each attribute and each pointer in the system occupies 100 Bytes, each page has space for 4.000 Bytes, the last names of customers are equally distributed on the first letter over the 26 letters of the alphabet, and the most frequent query on Customer asks for all customers whose last name falls into a given range.

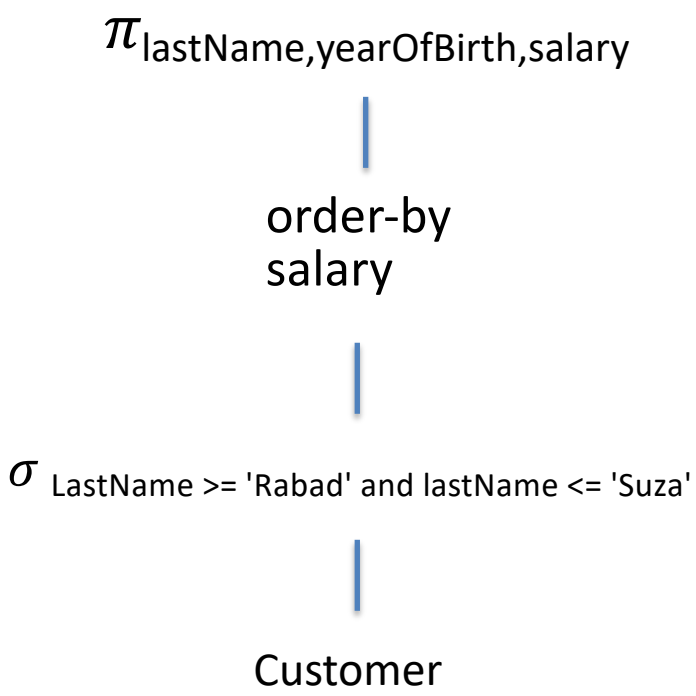
1. Tell which method would you use to store the relation.
2. Tell which logical plan and physical plan would you use to answer the following query Q

```
select lastName, yearOfBirth, salary
from Customer
where lastName >= 'Rabad' and lastName <= 'Suza'
order by salary
```

3. Assuming that 65 free buffer frames are available, tell which is the cost of executing the physical query plan you have defined for item 2, in terms of the number of page accesses.

Solution exercise 7

The logical plan associated to the query code is as follows:



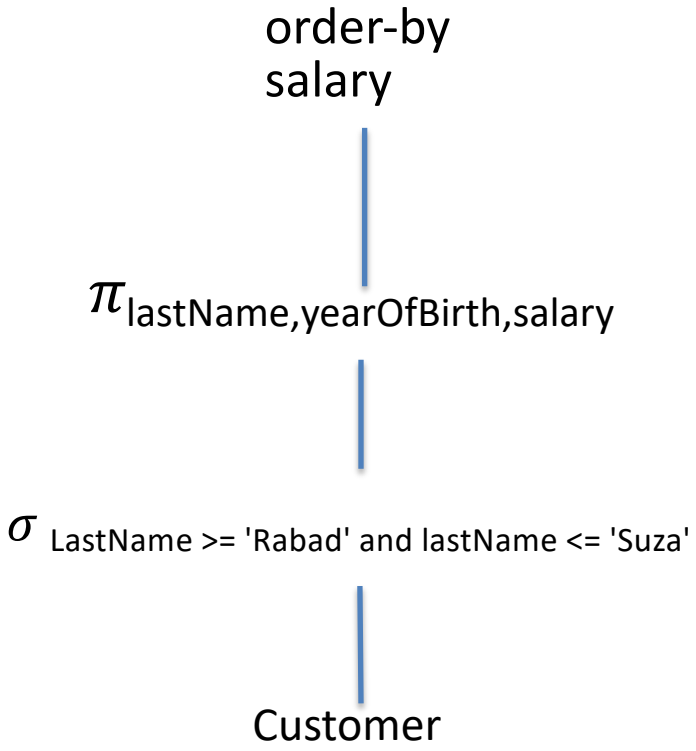
Since the most frequent query on Customer asks for all customers whose last name falls into a given range, we focus our attention on the method that stores the relation in a sorted file with sorting key lastName, with an associated clustering, sparse tree-based index on search key lastName. Indeed, it is well known that range queries are well supported by a clustering B+-tree index.

Solution exercise 7

- Having the clustering B+-tree index, we can use the index for the “selection” operator, thus finding the first page of Customer with the tuples satisfying the where condition. We then exploit the fact that the index is clustering, and sort on salary all pages with tuples satisfying the condition, so as to get the final result. Note that we do not materialize the result of the “selection” operator. Rather, a pipeline approach is used to pass the result of the “selection” operator to the sorting operator.
- Note that it seems promising to push projection under the order by, because the size of the operand will decrease.
- Note also that we cannot push projection under selection, because otherwise we cannot use the index.

Solution exercise 7

The logical plan of query Q is now as follows:



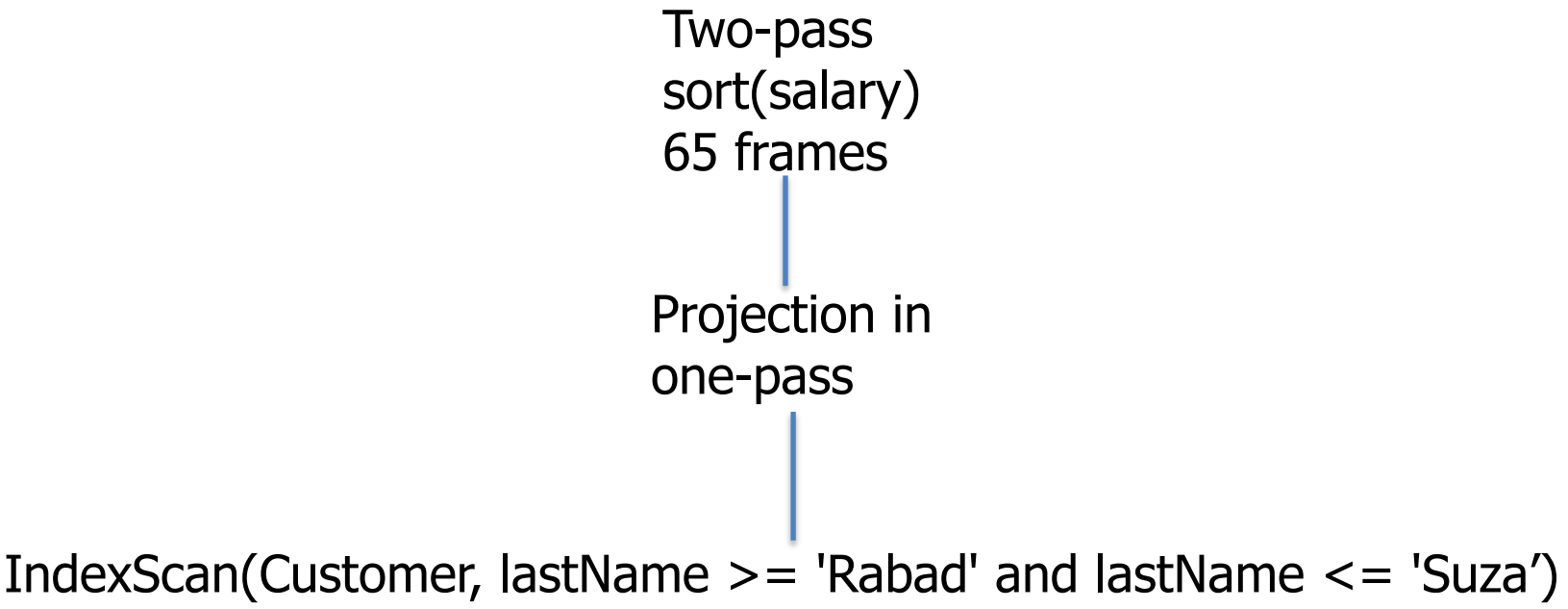
Let us now choose the physical query plan.

Solution exercise 7

- Algorithm for selection. Since each value or pointer occupies 100 Bytes, every tuple has 4 values, and since each page has space for 4000 Byte, it follows that every tuple occupies 400 Bytes, which means that each page has space for 10 tuples of Customer, and the relation is stored in 65.000 pages. Also, we have that each data entry requires 200 Bytes, and each page has space for 20 data entries. Taking into account the 67% occupancy rule for the leaves of the tree-based index, we have that each leaf contains 13 data entries. Also, since each page has space for 20 index entries, we can assume the value 15 for the fan-out. Note that, since the index is sparse, it stores one value of lastName for each page of Customer, and therefore the number of leaves are 65.000 / 13 = 5.000. It follows that reaching the right leaf requires $\log_{15} 5.000 = 4$ page accesses.
- Projection. After reaching the right leaf, we then have to compute the projection of three over 4 attributes. Since we have to access the tuples with last names starting with two letters ('R' and 'S'), and we know that the last names of customers are equally distributed on the first letter over the 26 letters of the alphabet, the number of qualifying records are (650.000/26) × 2 = 50.000, stored in 50.000/10 + 1 =5.001 pages. Since we choose ¾ of attributes, we count (3/4) × 5.001= 3.751 pages to pass to the sorting operator.
- Sorting. The records of such pages must be sorted having 65 buffer frames available. Since 3.751 < 65 × 64 (i.e., 4.160), we can use the two-pass sorting algorithm, with a cost of 3 × 3.751 = 11.253 page accesses.

Solution exercise 7

Here is the very simple physical query plan:



Notice that if we had not pushed projection under order-by, we could have not used a two-pass algorithm for sorting, because 5.001 > 4160.

Exercise 8

Assume we have the relations

R(A,B,C,D)

S(E,F,G,H,I,L)

R occupies 1.500 pages, with 20 tuples per page, and has a dense clustering tree-based index on A using alternative 2.

S occupies 4.800 pages.

We have to answer the following query:

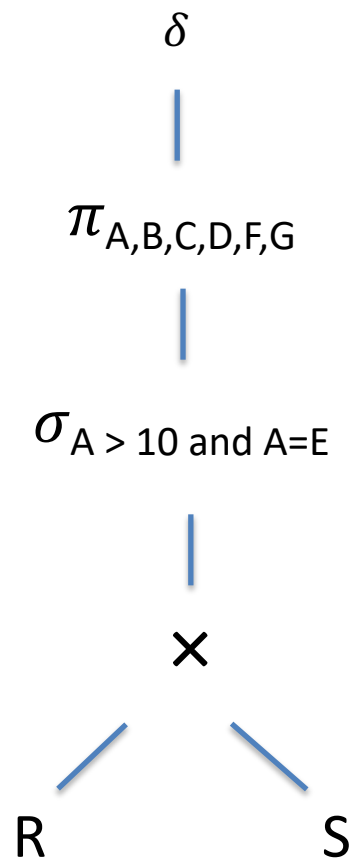
```
select distinct A,B,C,D,F,G
from R, S
where A > 10 and A=E
```

Under the assumption that we have 50 free frames in the buffer.

Solution exercise 8

Here is the logical plan associated

with the query code:

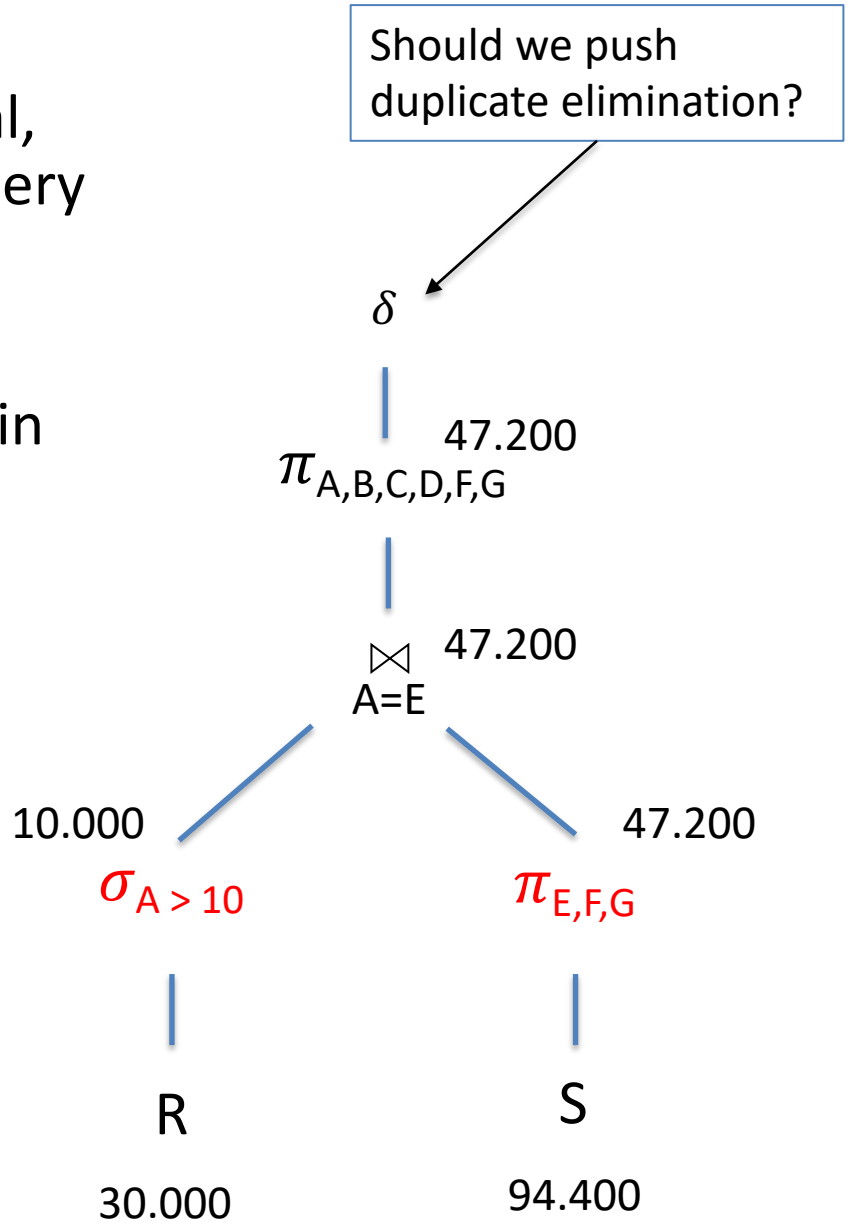



```
select distinct A,B,C,D,F,G
from R, S
where A > 10 and A=E
```

Solution exercise 8

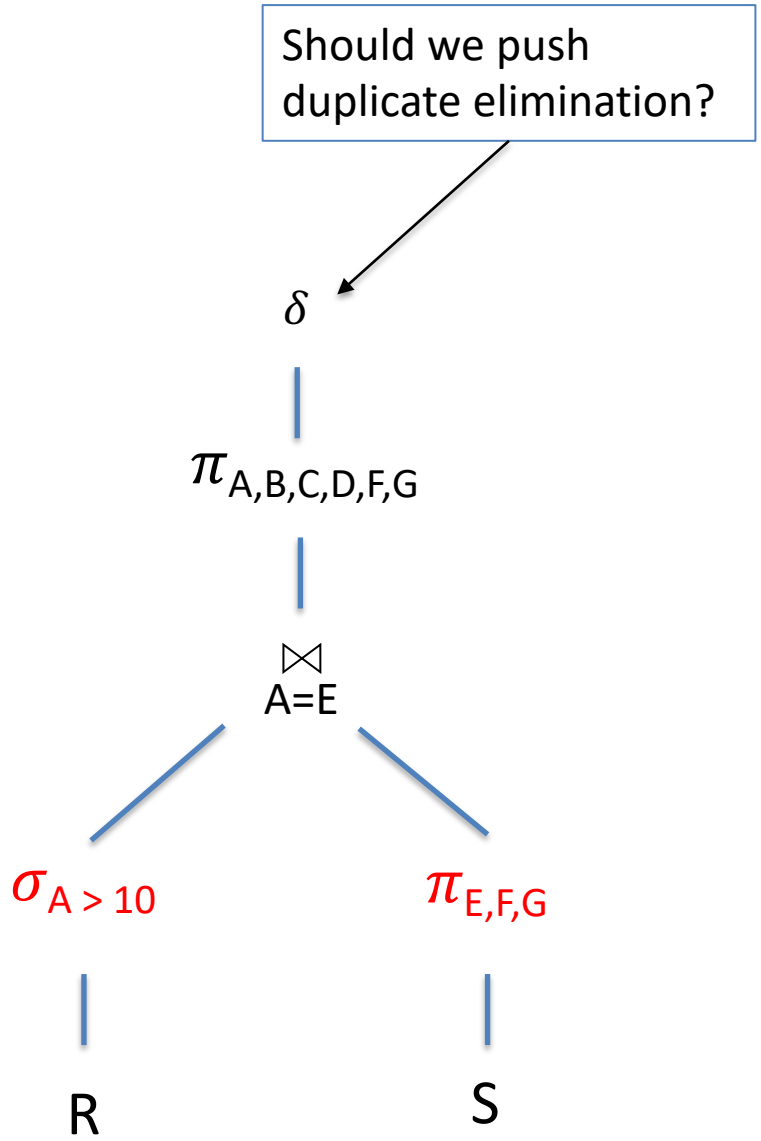
Since pushing selection and, in this case, pushing projection is obviously beneficial, we immediately transform the logical query plan as follows:

Note that, since 20 tuples of 4 values fit in one page, we assume that 13 tuples of 6 values fit in one page. Therefore, we assume that S has 94.400 tuples.



select distinct A,B,C,D,F,G
from R, S
where A > 10 and A=E

Solution exercise 8



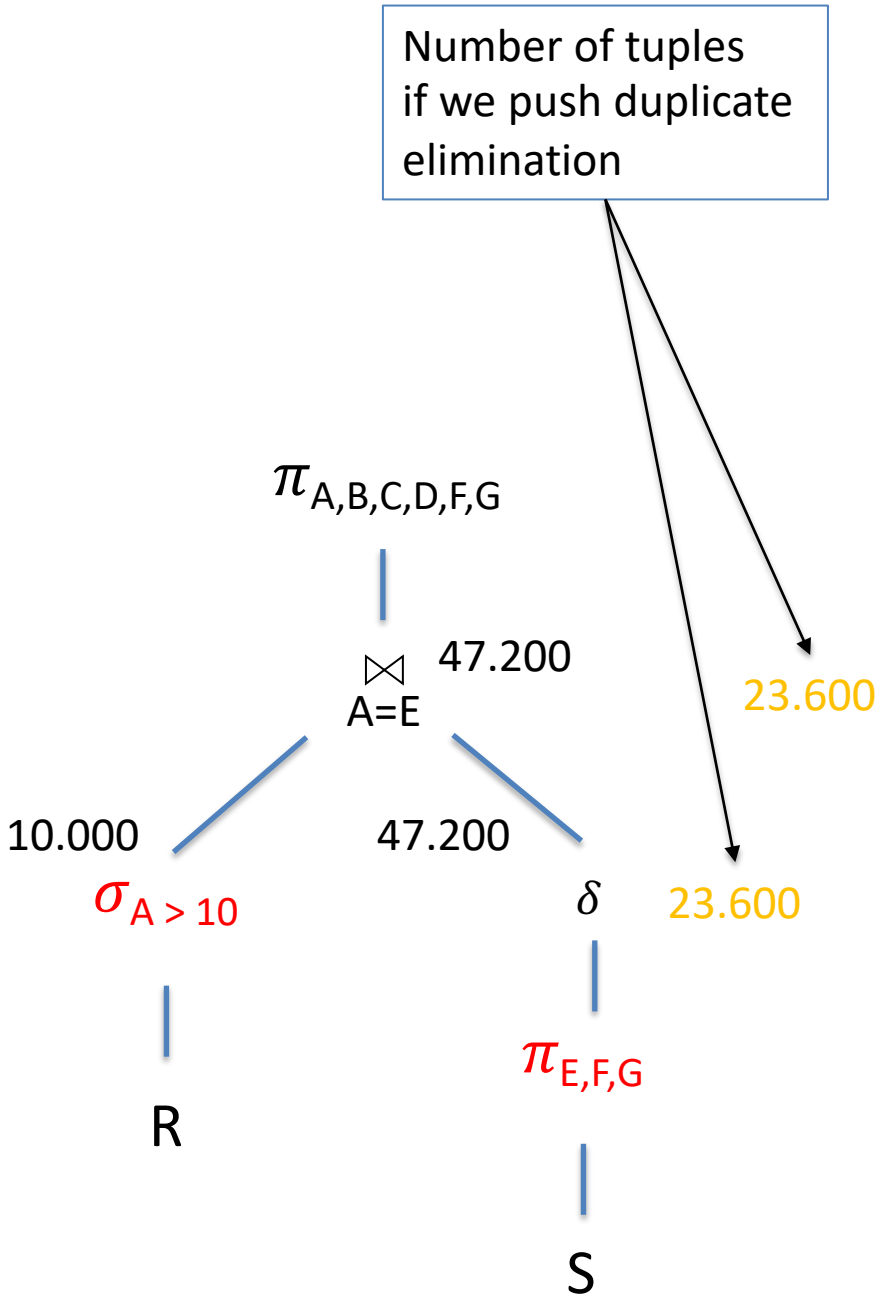
select distinct A,B,C,D,F,G

from R, S

where A > 10 and A=E

Solution exercise 8

If we remain at the logical level, it seems that pushing duplicate elimination is beneficial (note that pushing duplicate elimination on R is useless).



select distinct A,B,C,D,F,G
from R, S
where A > 10 and A=E

Solution exercise 8

But let us focus on physical query plan.

First of all, the selection operator will be done through the index, and therefore, let us compute the cost of accessing the index.

R occupies 1.500 pages, with 20 tuples per page, and the associated index is a dense clustering tree-based index on A using alternative 2. Note that R has 30.000 tuples.

Since 20 tuples of 4 values fit in one page, 40 data entries (of 2 values) fit in one page. Since we have to store 30.000 values, the leaves of the tree require 30.000/40=750 pages, and taking into account the 67% occupancy rule, we have 1.125 leaves.

Also, we can assume 30 as fan out, and therefore, the cost of accessing the index with a specific value for the search key is $\log_{30} 1.125 = 3$.

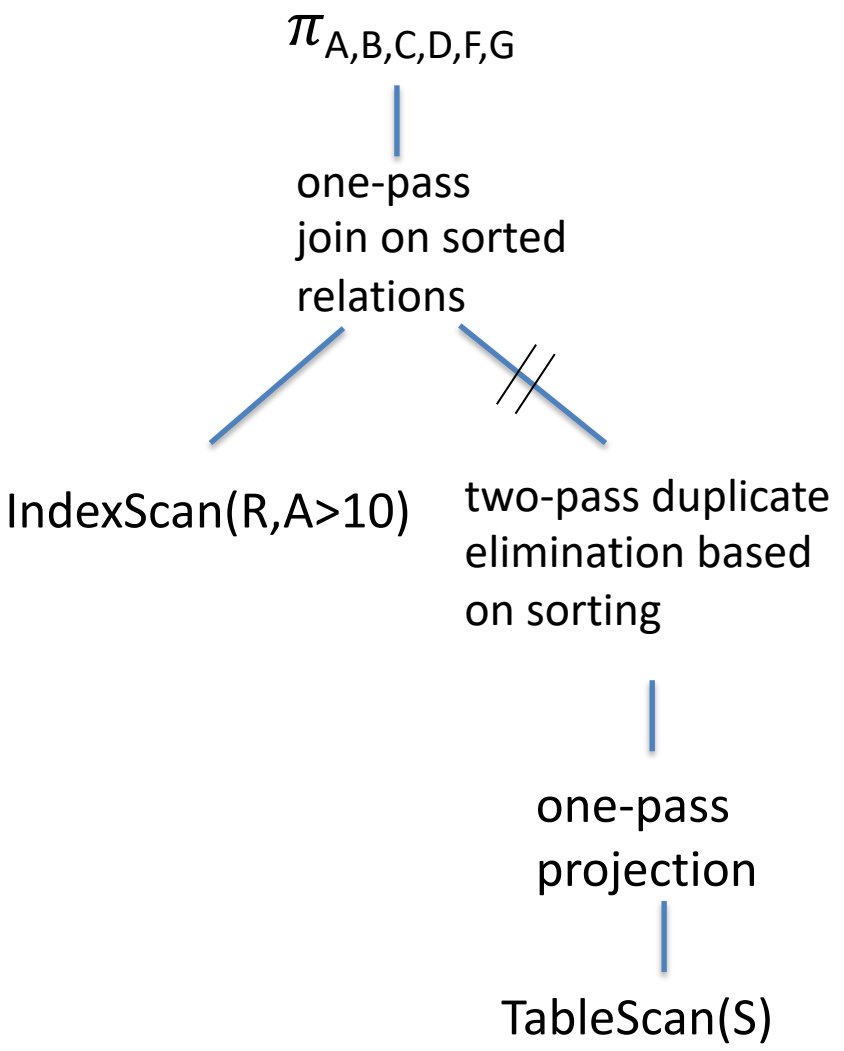
Solution exercise 8

If we consider the physical plan, we realize that, if we push duplicate elimination, the corresponding cost in terms of page accesses is

4.800 + 2 × 2.400 + 1.200 + 753 + 1.200 = 13.503

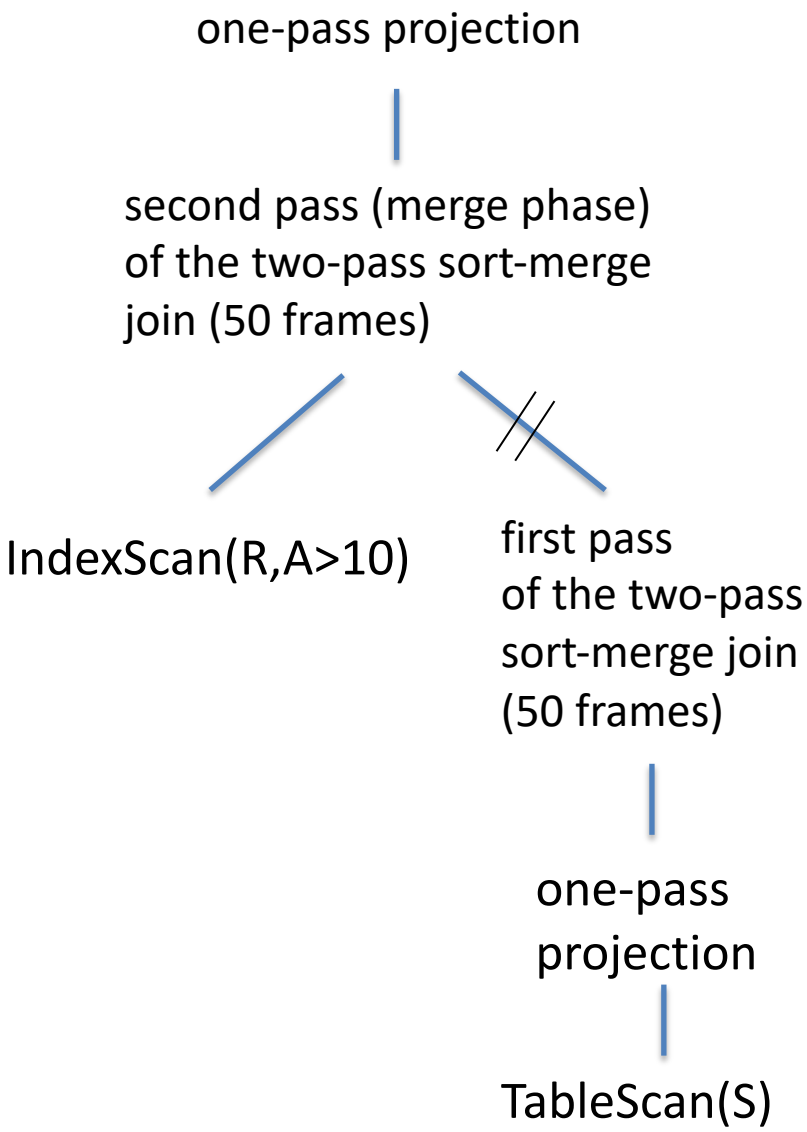
where:

- 4.800 is the cost of table scan of S
- 2 × 2.400 is the cost of duplicate elimination, i.e., writing of sublists and merge of sublists for the 4.800/2 pages of the projection of S
- 1.200 pages is the cost of writing the result of duplicate elimination (which we consider half of the input)
- 753 + 1.200 is the cost of the one-pass join: 753 is the cost of reading the first operand (where 750 is the number of pages satisfying A > 10, and 3 is the access to the index), and 1.200 is the number of pages of the second operand



Solution exercise 8

On the other hand, if we do not push duplicate elimination, the physical plan is the one shown here:



The corresponding cost in terms of page accesses is

$4.800 + 2 \times 2.400 + 1.503 + 1.200 = 12.303$

where

- 4.800 is the cost of table scan of S
- 2×2.400 is the first pass of the two-pass sort-merge join (reading and writing 2.400/50 = 48) for the 4.800/2 pages of the projection of S
- 1.503 + 1.200 is the cost of the second pass (merge phase) of the two-pass sort-merge join, taking into account that we have the relation coming out from the index scan already sorted so, 49 sorted relations and merged using 50 free frames
- one-pass join: 1.503 is the cost of reading the first operand (where 3 is the access to the index), and 1.200 is the number of pages of the second operand

We conclude that this is the best physical query plan.