Query Processing

Exercise Session 3

I/O Cost When Selecting from Relations on the Disk

Main Topic

- We deal with the selection operator and address two issues
  - Estimating the size of the result, which is needed in order to determine the cost of subsequent operations
  - Finding the best way to evaluate a selection that is applied to a base relation and computing its I/O cost

Basic Parameters

- $B(R)$ is the number of blocks of a relation R
  - $B(R)$ is the number of blocks that R actually occupies on the disk (blocks on the disk are not necessarily full)
- $T(R)$ is the number of records in R
- $V(R,L)$ is the number of distinct values for the attributes of L in the relation R, i.e., the number of distinct records in $\pi_L(R)$
- Hence, $T(R)/V(R,L)$ is the average number of records with the same value for the attributes of L
- Give a sufficient condition for $T(R) = V(R,L)$

Are Blocks Full or Not?

- Let n be the average number of records of R per block, and let m be the maximal number of records of R that can fit in one block
  - $n = T(R)/B(R)$
  - n/m is the fraction of space used in each block for records of R
    - The rest is either free space or used for another relation
- Unless otherwise stated, we assume that $n = m$
- For intermediate results, $n = m$

We assume that the constant *a* appears in column *A*

$$\sigma_{A=a}(R)$$

Blocks of results are always full

- What is the size of the result?
  - Number of records in the result is *T(R)/ V(R,A)*
  - Number of blocks in the result is $\lceil B(R)/V(R,A) \rceil$ provided that blocks of *R* are full
    - True because the result is a subset of *R*
    - On the disk, however, the records of *R* that are in the result may be spread over a much larger number of blocks (if *R* is not clustered on *A*)
  - If blocks of *R* are only 80% full,
    - then the number of blocks in the result is $\lceil 0.8B(R)/V(R,A) \rceil$
- No need to add +1 to either $\lceil B(R)/V(R,A) \rceil$ or $\lceil 0.8B(R)/V(R,A) \rceil$ for the worst case – why?

Intuitive Example

- To better understand what is going on, think of boxes filled with red and green apples
  - There is a total of 50 boxes
  - Only 10% of the apples are red
- Suppose that the boxes are completely full
  - If we retrieve the red apples, we need only 5 boxes to store them separately
  - But we may have to access all 50 boxes to retrieve the red apples if they are not clustered
- If the boxes are only 80% full, then 4 boxes are sufficient for storing the red apples separately, but we may still have to access all 50 boxes in order to retrieve all the red apples

Comments

- Only in the case of selection, can we easily express the number of blocks of the result in terms of *B(R)*
  - For other operations, we find the number of blocks of the result from
    - estimating the number of records in the result, and
    - calculating how many records fit in one block
- Generally, we need to know the size of the result in order to compute the I/O cost of subsequent operations

Example

- This example shows that it is impossible to compute the number of blocks of the Cartesian product $R \times S$ from $B(R)$ and $B(S)$
- Suppose that $B(R) = B(S) = 1$ and consider two cases
  - First, each relation has 1 record,
  - Second, each relation has 100 records
  - In both cases, a block can store 100 records of the result
- In the first case, the result has 1 block and in the second – 100 blocks

We assume that the constant *a* appears in column *A*

$$\sigma_{A=a}(R)$$

Add "+1" for the worst case (not essential, since the difference is small)

- What is the I/O cost? (*R* is in the DB – on disk)
- Three possible ways of computing
  - Scan all of *R* and apply the selection as a filter
    - I/O cost is *B(R)*
  - Use a clustering index on *A*
    - I/O cost is $\lceil B(R)/V(R,A) \rceil$
      - » regardless of whether blocks of *R* are full or not (*B(R)* is the size of *R* on the disk)
  - Use a non-clustering index on *A*
    - I/O cost is *T(R)/ V(R,A)* (in the worst case)
    - We always take the worst case, because it is considerably larger than the best case
    - If *T(R)/ V(R,A)* > *B(R)*, then it is better to scan *R* without using the index

I/O cost of writing the output not included

The cost of accessing the index is not included

Comment

- When R is clustered on A, the same fraction $1/V(R,A)$ of records and blocks of R are relevant to the selection, regardless of whether blocks are full or not
- The worst case $T(R)/V(R,A)$ can actually happen when using a non-clustering index even if $T(R)/V(R,A) > B(R)$

The Size of $\sigma_{A>a}(R)$

- The result has $\lceil T(R)/3 \rceil$ records
- The result has $\lceil B(R)/3 \rceil$ blocks, provided that blocks of R are full
- If blocks of R are 90% full, the number of blocks in the result is $\lceil 0.9B(R)/3 \rceil$
- Similarly, for $A < a$

I/O Cost of Computing $\sigma_{A>a}(R)$

Similarly, for $A < a$

Three options:

- Scan *R* at an I/O cost of *B(R)*
- *R* is sorted on *A* and there is an index on *A*
  - Use the index to find the first record with $A>a$ and scan the file from that point
    - Blocks of a file are chained in both directions
    - I/O cost is $\lceil B(R)/3 \rceil$ (i.e., same proportion as that of the number of records satisfying the selection)
      - regardless of whether blocks of *R* are full or not
- *R* is not sorted on *A* but there is a sorting index on *A*
  - I/O cost is $\lceil T(R)/3 \rceil$ which is most likely > *B(R)*

How many times do we use the index?

Comments

- The I/O costs in the previous slides do not include the cost of using the index
- For $\sigma_{A=a}(R)$
  - If the index is clustering, we use it once
  - Otherwise, the index must be dense and we use its lowest level for each record in the result
- For $\sigma_{A>a}(R)$
  - If the file is sorted, we use the index just once
  - If the index is sorting but the file is not sorted, then the index must be dense and we use the lowest level of the index for each record of the result

We assume that blocks in the lowest level of an index, as well as blocks of the file, are chained in both directions

Comments (continued)

- Thus, the third option (in Slides 10 and 13) could actually be worse than it appears to be when ignoring the I/O cost of using the index
  - If the number of records that satisfy the selection is relatively small, then we may need only one block from the lowest level of the dense index
- We could lower the I/O cost of the third option (in Slides 10 and 13) if we have a large buffer so that only a small number of blocks (of the relation) are read more than once

I/O Cost of $\sigma_{A=a \wedge B>b}(R)$

- Best if *R* is lexicographically sorted on *A,B* and there is an index on *A,B*
  - We use the index once and then scan the file
- Index only on *A* – I/O cost same as $\sigma_{A=a}(R)$
- Index only on *B* – I/O cost same as $\sigma_{B>b}(R)$
- No index – scan all of *R* at a cost of *B(R)*
- Index according to lexicographic order *B,A*
  - Use it just as an index on *B*

In the last four cases, some of the comparisons are used as a filter

I/O Cost of $\sigma_{A=a \wedge B>b}(R)$ in the Best Case

- Recall that the number of records in the result is obtained by assuming independence, namely, it is $T(R)/(3V(R,A))$
- Since the file is lexicographically sorted on *A,B*
  - The same fraction of records and blocks of *R* are relevant to the selection
  - Thus, we have to access $\lceil B(R)/(3V(R,A)) \rceil$ blocks on the disk
    - regardless of whether the blocks of *R* are full or not

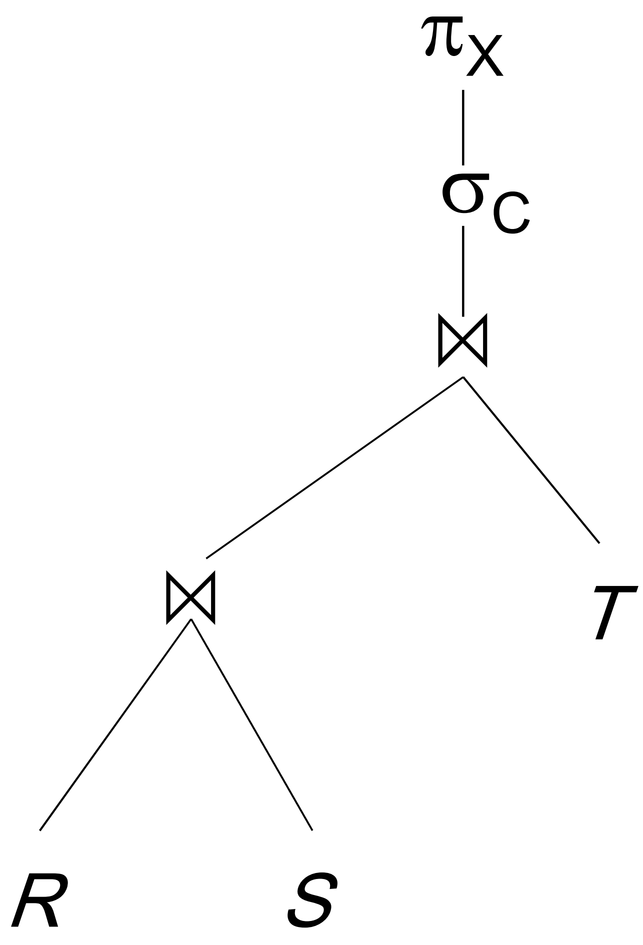
I/O Cost of $\sigma_{A=a \vee B>b}(R)$

- To avoid scanning all of R, we need two indexes
  - one on A and another on B
- Thus, the I/O cost is the sum of the costs of $\sigma_{A=a}(R)$ and $\sigma_{B>b}(R)$
- We must remove duplicates (even according to SQL semantics), and we can do it efficiently
  - When using the second index, check that the records do not satisfy the first condition

Dealing with Projections and Selections in Query Evaluation

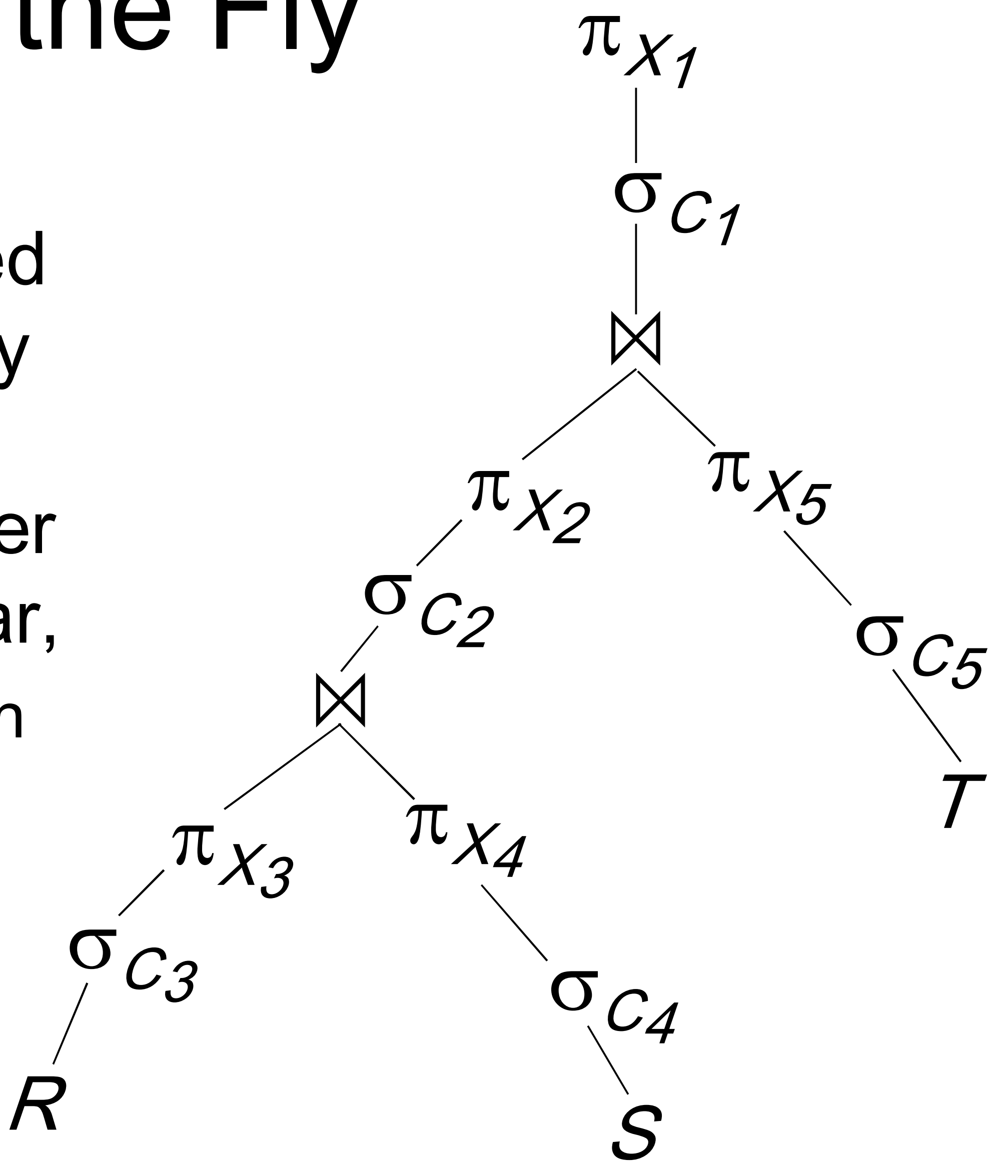
Pushing Projections and Selections

- Projections and selections should be pushed down as much as possible
- See details on how to do it in Slides 98-107 of the lecture on relational algebra, and Slide 40 of QP1.3_Tirgul



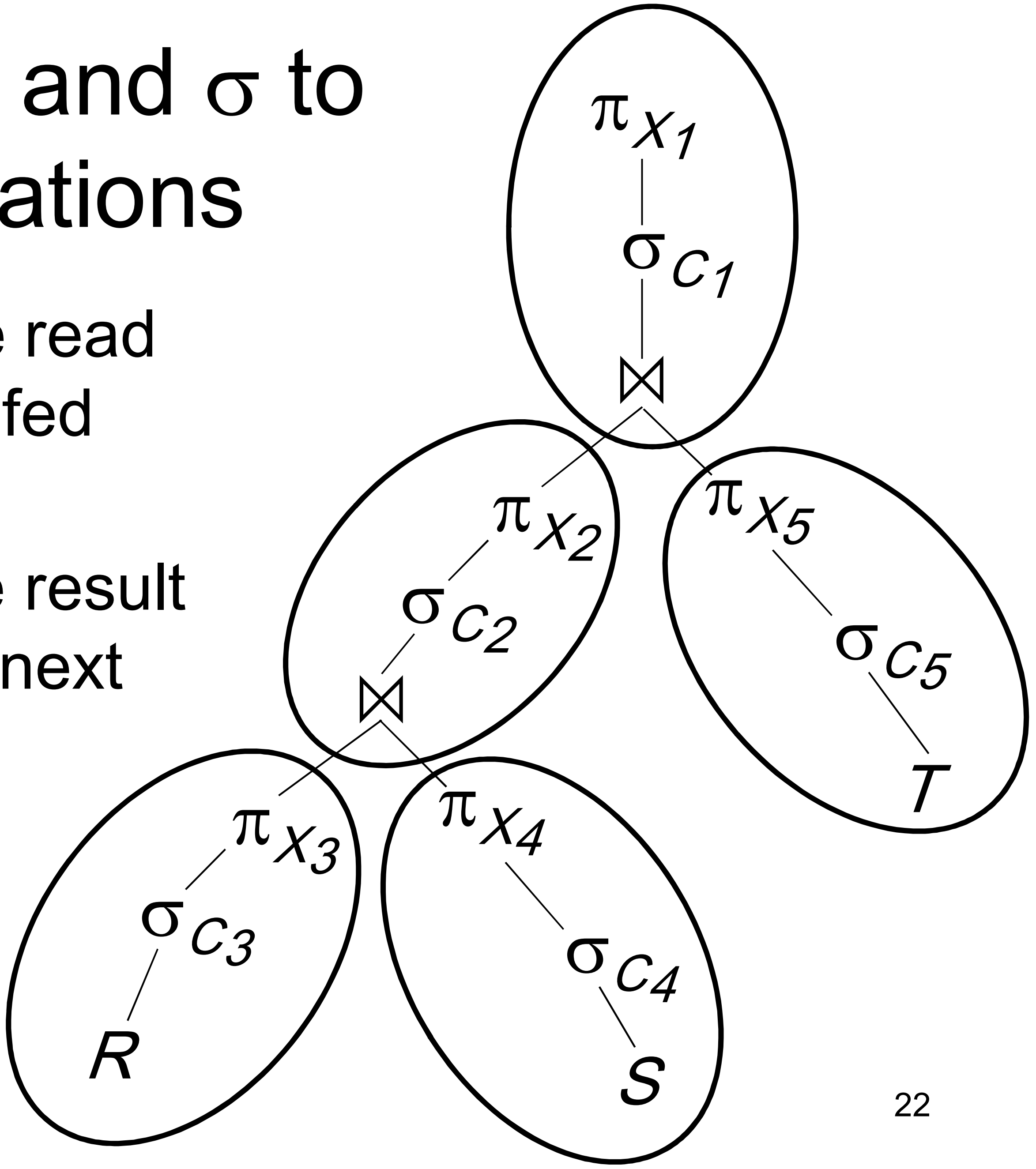
On the Fly

- Projections and selections are executed *on the fly*, namely, they are done in main memory as part of other operations; in particular,
  - When reading a relation from the database, and
  - When generating the result of a join



Attaching π and σ to Other Operations

- 3 relations are read from disk and fed into a join
- 1 intermediate result is fed into the next join
- 1 final result



Taking Selection & Projection into Account

- Selections and indexes determine the cost of the red ovals (on the previous slide), namely, the cost of reading relations from the disk
- Selections and projections affect the size of the result of each oval
- Selections and projections do not have any I/O cost in and of themselves

Comment

- We can find the selections and projections that can be pushed to the base relations before we find the optimal join order
- The projections and selections that can be pushed to intermediate results (but not to the leaves) have to be determined separately for each join order