Query Processing and Advanced Queries

Query Optimization (2)

Introduction

- We have optimized the logical query plan, applying relational algebra equivalences.
- In order to refine this plan into a physical query plan, we in particular need to choose one of the available algorithms to implement the basic operations (selection, join, . . .) of the query plan.
  - For each alternative physical query plan, we estimate its cost.
  - The cost estimates are based on the size estimates that we discussed in the previous chapter.

Introduction

- Disk I/O (read / write of a disk block) is orders of magnitude more expensive than CPU operations.
- Therefore, we use the *number of disk I/Os* to measure the cost of a physical query plan.
- We ignore CPU costs, timing effects, and double buffering requirements.
- We assume that the arguments of an operator are found on disk, but the result of the operator is left in main memory.

Introduction

- We use the following *parameters* (statistics) to express the cost of an operator:
  - B(R) = # of blocks containing R tuples,
  - f(R) = max # of tuples of R per block,
  - M = # memory blocks available in the buffer,
  - HT(i) = # levels in index i,
  - LB(i) = # of leaf blocks in index i.
- M may comprise the entire main memory, but typically the main memory needs to be shared with other processes, and M is much (!) smaller.

Introduction

- The performance of relational operators depends on many parameters such as the following ones.
  - Are the tuples of a relation stored physically contiguous (*clustered*)? If yes, the number of blocks to be read is greatly reduced compared to non-clustered storage.
  - Is a relation *sorted* by the relevant (selection, join) attribute? Otherwise, it may need to be sorted on-the-fly.
  - Which *indexes* exist? Some algorithms require the existence of a corresponding index.

Introduction

- Each operator (selection, join, . . .) in a logical query plan can be implemented by one of a fairly large number of alternative algorithms .
- We distinguish three types of algorithms:
  - *sorting-based* algorithms,
  - *hash-based* algorithms,
  - *index-based* algorithms.
- Sorting, building of hash table or building of index can either have happened in advance or may happen on the fly.

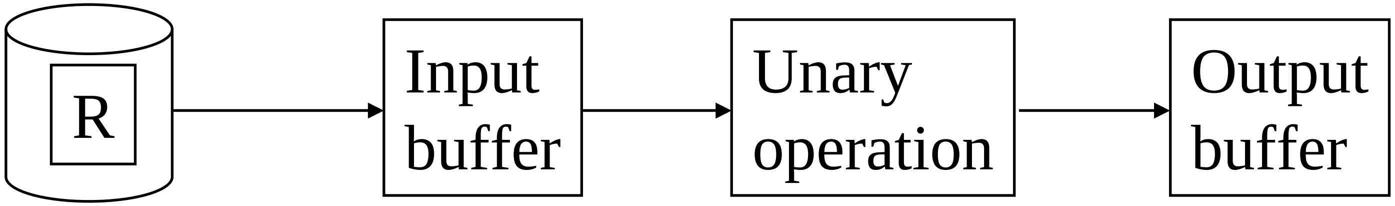
Introduction

- We can also categorize algorithms according to the number of passes over the data:

  - *one-pass algorithms*
    read data only once from disk,
  - *two-pass algorithms*
    read data once from disk, write intermediate relation back to disk and then read the intermediate relation once.
  - *multiple-pass algorithms*
    perform more than two passes over the data, not considered in class.

One-Pass Algorithms for Unary Operations

- Consider the unary, tuple-at-a-time operations, **selection** and **projection** on relation *R*.
- Read all the blocks of *R* into the *input buffer*, one at a time.
- Perform the operation on each tuple and move the selected / projected tuple to the *output buffer*.



One-Pass Algorithms for Unary Operations

- Output buffer may be input buffer of other operation and is not counted.
- Thus, algorithm requires only $M = 1$ buffer blocks.
- I/O cost is B(R).
- If some index is applicable for a selection, have to read only blocks that contain qualifying tuples.

One-Pass Algorithms for Binary Operations

- Binary operations: union, intersection, difference, Cartesian product, and join.
- Use subscripts B and S to distinguish between the set- and bag- version, e.g. $\cup_B$ and $\cup_S$.
- The *bag union* $R \cup_B S$ can be computed using a very simple one-pass algorithm: copy each tuple of *R* to the output, and copy each tuple of *S* to the output. (for the SUM model of bag union)
- I/O cost is B(R) + B(S), M = 1.

One-Pass Algorithms for Binary Operations

- Other binary operations require the reading of the smaller of the two input relations into main memory.
- One buffer to read blocks of the larger relation, M-1 buffers for holding the entire smaller table.
- I/O cost is B(R) + B(S).
- In main memory, a data structure is built that efficiently supports insertions and searches.
- Data structure, e.g., hash table or binary balanced tree. Space overhead can be neglected.
- $M > \min(B(R), B(S))$.

One-Pass Algorithms for Binary Operations

- For set union, read the smaller relation (S) into M-1 buffers, representing it in a data structure whose search key consists of all attributes.
- All these tuples are also copied to the output.
- Read all blocks of R into the M-th buffer, one at a time.
- For each tuple *t* of R, check whether *t* is in S. If not, copy *t* to the output.
- For set intersection, copy *t* to output if it also is in S.