Comp 5311 Database Management Systems

11. Join Algorithms Exercises
Join Algorithms - Example

- Consider the following two tables:
  Sailors\((\text{Sid, Name, Rating, Age})\)
  Reserves\((\text{Sid, Bid, Date})\)
- Each attribute (and pointer whenever applicable) is 20 bytes. Each page is 1000 bytes. There are 10,000 sailors, 40,000 reservations and a buffer of \(M=100\) pages.
- Assume the query: Find the names of sailors who have reservations
  \[\text{SELECT Name} \]
  \[\text{FROM Sailors, Reserves} \]
  \[\text{WHERE Sailors.Sid=Reserves.Sid} \]
- Since there are 10K sailors and 40K reservations, a sailor has on the average 4 reservations (but it is possible that some sailors have more, while others have none).
- Each sailor record is 80 bytes and each reserves record 60 bytes. There are 1000/80=12 sailors per page and 1000/60=16 reservations per page. Therefore, sailors contains 10000/12=834 pages and reserves 40000/16=2500 pages.
Block nested loops

- Sailors as outer: we read blocks of 98 pages of sailors at a time (there are 9 blocks of sailors). For each block we scan Reserves (page-by-page) to find matching tuples. One extra page is allocated for output buffer. The total cost is: $834 + 9 \times 2500 = 23,334$

- Reserves as outer: we read blocks of 98 pages of reserves at a time. For each block we scan Sailors to find matching tuples. There are $2500/98 = 26$ blocks of reserves. Total cost = $2500 + 26 \times 834 = 24,184$
Index Nested Loop (assume hash index on Reserves.Sid)

- For each sailor find the corresponding entry in the hash index on Reserves.Sid. We assume that this takes 1.2 page accesses per record of sailors (because of overflow buckets).
- Since each Sailor has on the average 4 reservations, and the hash index is non-clustering (secondary) we expect each Sid to have 4 matching records in Reserves. Therefore, we need 5.2 page accesses per record and the total cost is:
  - cost of reading Sailors + #records in sailors*5.2 = 834 + 10,000*5.2 = 52,834
- bad solution
Good solution

Notice that since we do not care about the boats that a sailor has reserved we can do an index-only scan— we do not need to retrieve any tuples from reserves— if a Sid exists in the index it means that a sailor has at least one reservation). Thus the total cost is: $834 + 10,000 \times 1.2 = 12,834$

Questions:
1] What if the above query requested also the Bids?
2] What if there were an additional condition on sailors "e.g., WHERE Sailor.Rating=8".
Sort Merge Join

- Sort Sailors on Sid

At each sorted run we read on 100 sailor pages but we only write 50 because we discard attributes Rating, Age (they are not needed for the join and are not required in the result)

Cost: $834 + 417 \text{ (pass 0)} + 417 + 417 \text{ (pass 1)} = 2085$
Sort Merge Join (cont)

• Sort Reserves on Sid

At each sorted run we read on 100 reserves pages but we only write 34 because we discard attributes Bid, Date (they are not needed for the join and are not required in the result) - so we assume that we only need $\frac{1}{3}$ of the space

Cost: $2500 + 850 = 4200$. As each sorted page of Reserves is generated we can directly find the joining tuples of sailors (in order to avoid writing the result of pass 1 in a temporary file and reading it again for the merge phase). Thus, for merging we only need to scan the sorted sailors file.

• Total cost: $2085 + 4200 + 417 = 6702$
Sort Merge Join (cont)

- Considering the relatively large buffer that we have, we could further improve the performance by sorting the two files concurrently. That is, we perform the pass 0 of Sailors (and create 9 sorted runs). Then pass 0 of Reserves (and create 25 sorted runs).

- Then we sort the two files at the same time; i.e., we allocate 9 buffer pages for the sorted runs of Sailors and 25 pages for reserves. Once the first sorted page is created during pass 1 of Sailors, the algorithm will proceed by sorting Reserves and creating the first sorted page. The two pages can be immediately matched without the need to materialize any intermediate file.

- When one of the two pages is exhausted (all its records have been matched) the algorithm proceeds by generating the second sorted page for this file and so on. This method improves the previous one by avoiding the cost of writing and reading the sorted sailors table in a temporary file, i.e., the total cost is: $6702 - (2 \times 417) = 5868$. 
• Assumptions: \( b_r \), \( b_s \) sizes of files (in pages), \( M \) main memory pages available
• Cost = 3(\( b_r + b_s \))
• \( k_1 = \lceil \frac{b_r}{M} \rceil \), \( k_2 = \lceil \frac{b_s}{M} \rceil \)
• In order to be able to sort both files with just two passes it should be: \( M > k_1 + k_2 \)  \( \Rightarrow \ M > \sqrt{b_r + b_s} \)
Hash Join (non-optimized)

• Use the small relation (Sailors = 834) as the build input.
• The number of buckets $n$ that we choose should be such that each partition for Sailors fits in memory, e.g., we can use 10 buckets (so that average bucket size is 83.4 pages).
• We first partition Sailors with cost = 834 + 834.
• Then we read and partition Reserves. Cost = 2500 + 2500. Note that each bucket of Reserves occupies more pages (250) than the available memory, but we don't care (only the buckets of Sailors should fit in memory).
• Finally we read each bucket of Sailors and match it against the corresponding bucket of Reserves. We output the matching tuples to the user. Cost = 834 + 2500.
• Total Cost = 3(2500 + 834) = 10,002
Hybrid Hash–Join

- **Main feature of hybrid hash join:** Keep the first partition(s) of the build relation in memory.
- We again use Sailors as the build input. Let's say we assign 90 memory pages to the first partition/bucket, so that we keep it all in memory. Cost = 834 + (834 - 84) (we do not write the first bucket to memory)
- We read Reserves page by page. Each record that belongs to bucket 1, is matched directly with the first bucket of Sailors. The other tuples go to the corresponding buckets and are written back to the disk. Cost = 2500 + (2500 - 250).
- Finally we read buckets 2-10 of Sailors and match them to the corresponding buckets of Reserves. Cost = (834 - 84) + (2500 - 250).
- Total cost = 3(2500 + 834) - 2(84 + 250) = 10,002 - 668 = 9,334
Optimized Hybrid Hash Join

- **Main idea:** Remove extra attributes (similar to sort-merge join).
- When we partition Sailors, we discard the rating and age attributes since they are not needed for the query. The remaining file is 417 pages and each bucket of Sailors consumes 42 pages. Thus, we can keep the first 2 buckets in memory (84 pages). We also need 8 main memory pages for the remaining 8 pages.
- **Cost of partitioning sailors:** $834 + 417 - 84 = 1167$
- We read Reserves page by page. Each record that belongs to the first two buckets, is matched directly with the corresponding bucket of Sailors (which is in memory). The other tuples go to the corresponding buckets, after removing the useless attributes (bid, date).
- After removing bid, date, the Reserves file has 1/3 of its original size: 834 pages. Each bucket is on the average 84 pages. We have to write to the disk all buckets, except for the first two (168 pages):
- **Cost of partitioning reserves:** $2500 + 834 - 168 = 3166$
- Finally we read buckets 3-10 of Sailors and match them to the corresponding buckets of Reserves. Cost= $(417 - 84) + (834 - 168) = 999$. 
- **Total cost:** $1167 + 3166 + 999 = 5332$