Query Processing
Sort/Hash-based (Optional)

CS348 Spring 2023
Outline

• Scan
  • Selection, duplicate-preserving projection, nested-loop join

• Index
  • Selection, index nested-loop join, zig-zag join

• Sort
  • External merge sort, sort-merge join, union (set), difference, intersection, duplicate elimination, grouping and aggregation

• Hash (Optional)
Operators that benefit from sorting

• Union (set), difference, intersection
  • More or less like SMJ

• Duplication elimination
  • External merge sort
    • Eliminate duplicates in sort and merge

• Grouping and aggregation
  • External merge sort, by group-by columns
    • Trick: produce “partial” aggregate values in each run, and combine them during merge
      • This trick doesn’t always work though
        • Examples: SUM(DISTINCT ...), MEDIAN(...)
Outline

• Scan
  • Selection, duplicate-preserving projection, nested-loop join

• Index
  • Selection, index nested-loop join, zig-zag join

• Sort
  • External merge sort, sort-merge join, union (set), difference, intersection, duplicate elimination, grouping and aggregation

• Hash (Optional)
  • Hash join, union (set), difference, intersection, duplicate elimination, grouping and aggregation
Hashing-based algorithms

Hash join

\[ R \bowtie_{R.A=S.B} S \]

- **Main idea**
  - Partition \( R \) and \( S \) by hashing their join attributes, and then consider corresponding partitions of \( R \) and \( S \)
  - If \( r.A \) and \( s.B \) get hashed to different partitions, they don’t join

Nested-loop join considers all slots

Hash join considers only those along the diagonal!
Partitioning phase

- Partition $R$ and $S$ according to the same hash function on their join attributes

Each partition has a size of $\frac{B(R)}{(M-1)}$
Probing phase

• Read in each partition of \( R \), stream in the corresponding partition of \( S \), join
  • Typically build a hash table for the partition of \( R \)
    • Not the same hash function used for partition, of course!

Disk

\( R \) partitions

\( S \) partitions

Memory

For each \( S \) tuple, probe and join
Performance of (two-pass) hash join

• If hash join completes in two phases:
  • I/O’s: $3 \cdot (B(R) + B(S))$
    • 1st phase: read $B(R) + B(S)$ into memory to partition and write partitioned $B(R) + B(S)$ to disk
    • 2nd phase: read $B(R) + B(S)$ into memory to merge and join

• Memory requirement:
  • In the probing phase, we should have enough memory to fit one partition of $R$: $M - 1 > \frac{B(R)}{M-1}$
  • $M > \sqrt{B(R)} + 1$
  • We can always pick $R$ to be the smaller relation, so:
    $$M > \sqrt{\min(B(R), B(S))} + 1$$
Generalizing for larger inputs

• What if a partition is too large for memory?
  • Read it back in and partition it again!
  • Re-partition $O(\log_M B(R))$ times
Hash join versus SMJ

(Assuming two-pass)

• I/O’s: same

• Memory requirement: hash join is lower

\[ \sqrt{\min(B(R), B(S))} + 1 < \sqrt{B(R) + B(S)} \]

• Hash join wins when two relations have very different sizes

• Other factors
  • Hash join performance depends on the quality of the hash
    • Might not get evenly sized buckets
  • SMJ can be adapted for inequality join predicates
  • SMJ wins if \( R \) and/or \( S \) are already sorted
  • SMJ wins if the result needs to be in sorted order
What about nested-loop join?

- May be best if many tuples join
  - Example: non-equality joins that are not very selective

- Necessary for black-box predicates
  - Example: WHERE user_defined_pred(R. A, S. B)
Other hash-based algorithms

- Union (set), difference, intersection
  - More or less like hash join

- Duplicate elimination
  - Check for duplicates within each partition/bucket

- Grouping and aggregation
  - Apply the hash functions to the group-by columns
Summary of techniques

• Scan
  • Selection, duplicate-preserving projection, nested-loop join

• Index
  • Selection, index nested-loop join, zig-zag join

• Sort (Optional)
  • External merge sort, sort-merge join, union (set), difference, intersection, duplicate elimination, grouping and aggregation

• Hash (Optional)
  • Hash join, union (set), difference, intersection, duplicate elimination, grouping and aggregation
Another view of techniques

• Selection
  • Scan without index (linear search): $O(B(R))$
  • Scan with index – selection condition must be on search-key of index
    • B+ index: $O(\log(B(R)))$
    • Hash index: $O(1)$

• Projection
  • Without duplicate elimination: $O(B(R))$
  • With duplicate elimination
    • Sorting-based: $O(B(R) \cdot \log_M B(R))$
    • Hash-based: $O(B(R) + t)$ where $t$ is the result of the hashing phase

• Join
  • Block-based nested loop join (scan table): $O(B(R) \cdot \frac{B(S)}{M})$
  • Index nested loop join $O(B(R) + |R| \cdot (\text{index lookup}))$
  • Sort-merge join $O(B(R) \cdot \log_M B(R) + B(S) \cdot \log_M B(S))$
  • Hash join $O(B(R) \cdot \log_M B(R) + B(S) \cdot \log_M B(S))$