Query Processing II
Join Operation

Several different algorithms to implement joins

- Block nested-loop join
- Indexed nested-loop join
- Merge-join
- Hash-join
Block Nested-Loop Join

\[
\text{for each block } B_r \text{ of } r \text{ do begin }
\]
\[
\text{for each block } B_s \text{ of } s \text{ do begin }
\]
\[
\text{for each tuple } t_r \text{ in } B_r \text{ do begin }
\]
\[
\text{for each tuple } t_s \text{ in } B_s \text{ do begin }
\]
\[
\text{Check if } (t_r, t_s) \text{ satisfy the join condition }
\]
\[
\text{if they do, add } t_r \cdot t_s \text{ to the result. }
\]
\end{align*}

\[r : \text{outer relation}
\]

\[s : \text{inner relation}
\]

\[\text{cost } = b_r \cdot b_s + b_r,
\]
\[
\text{Assuming one buffer for each}
\]
Block Nested-Loop Join

r

s

...
Block Nested-Loop Join

- Requires no indices
- Can be used with any join condition
- cost = $b_r \times b_s + b_r$
  - Which relation shall be out relations?
Allocate More Buffers to Outer Loop

Cost?

r

s
Allocate More Buffers to Inner Loop

Cost?

How to allocate $M$ buffers to inner loop/out loop/result?
Indexed Nested-Loop Join

- Index available on the inner relation’s join attribute
  - Cost = $b_r + n_r \times c$
  - Where $c$ is the cost of traversing index and fetching all matching $s$ tuples for one tuple or $r$

- Index available on the outer relation’s join attribute
  - We do not use it, can we?
Indices available on join attributes of both \( r \) and \( s \)

Which one should be the out loop?
Exercise

- Compute $\text{depositor} \bowtie \text{customer}$, with $\text{depositor}$ as the outer relation.

- $\text{Customer}$ has a secondary B$^+$-tree index on $\text{customer-name}$
  - Blocking factor 20 keys

- $\#\text{customer} = 400b/10,000t$  $\#\text{depositor} = 100b/5,000t$

- Block nested loops
  - $400\times100 + 100 = 40,100$

- Indexed nested loops
  - $100 + 5000 \times 5 = 25,100$ disk accesses.
  - CPU cost likely to be less than that for block nested loops join
Merge-Join

1. Sort both relations on their join attribute (if not already sorted on the join attributes).
2. Merge the sorted relations to join them
   1. Join step is similar to the merge stage of the sort-merge algorithm.
   2. Every pair with same value on join attribute must be matched

Cost = \( br + bs + \) the cost of sorting if relations are unsorted
Merge-Join

But there is a problem!

We assume records with same value are in the same block!
Merge-Join

- Can be used only for equi-joins and natural joins
- Why not on inequality, greater than, less than joins?
Exercise

- Compute $\text{depositor} \bowtie \text{customer}$, with $\text{depositor}$ as the outer relation.
- $\text{Customer}$ has a secondary B$^+$-tree index on $\text{customer-name}$
  - Blocking factor 20 keys
- $\#\text{customer} = 400b/10,000t$ $\#\text{depositor} = 100b/5,000t$
- Merge join
  - $\log(400) + \log(100) + 400 + 100 = 516$
Hybrid Merge-join

- If one relation is sorted, and the other has a secondary $B^+$-tree index on the join attribute
  - Merge the sorted relation with the leaf entries of the $B^+$-tree.
  - Sort the result on the addresses of the unsorted relation’s tuples
  - Scan the unsorted relation in physical address order and merge with previous result, to replace addresses by the actual tuples
    - Sequential scan more efficient than random lookup
Hybrid Merge-join