Query Processing 3

Instructor: Beom Heyn Kim

beomheynkim@hanyang.ac.kr

Department of Computer Science

Overview

- Join Operation
- Assignments

Join Operation

- Join Operation:
  - Equi-join: $r \bowtie_{r.A=s.B} s$
  - Natural join: $r \bowtie s$
  - Theta join: $r \bowtie_{\theta} s$ for an arbitrary join condition θ

  Equality join condition
- Several different algorithms to implement joins
  - Nested-loop join
  - Block nested-loop join
  - Indexed nested-loop join
  - Merge-join
  - Hash-join
- Choice based on cost estimate
- Running example: *student* ⋈ *takes*
  - Assume:
    - Number of records of *student*: 5,000 *takes*: 10,000
    - Number of blocks of *student*: 100 *takes*: 400

Nested-Loop Join

- To compute the theta join $r \bowtie_\theta s$

  for each tuple t_r **in** r **do begin**

  for each tuple t_s **in** s **do begin**

  test pair (t_r, t_s) to see if they satisfy the join condition θ

  if they do, add $\boxed{t_r \bullet t_s}$ to the result.

  end

  end

$t_r \bullet t_s$ denotes the tuple constructed by concatenating the attribute values of tuples t_r and t_s

- r is called the **outer relation** and s the **inner relation** of the join.
- Requires no indices and can be used with any kind of join condition.
- Expensive since it examines every pair of tuples in the two relations.

Nested-Loop Join (Cont.)

- In the worst case, if there is enough memory only to hold one block of each relation, the estimated cost is

  $n_r * b_s + b_r$ block transfers, plus $\boxed{n_r} + b_r$ seeks

  n_r: Number of tuples in relation r

- If a relation fits entirely in memory, use that as the inner relation.
  - Reduces cost to $b_r + b_s$ block transfers and 2 seeks
- Assuming the worst case memory availability, cost estimate is
  - with *student* as outer relation:
    - 5000 * 400 + 100 = 2,000,100 block transfers,
    - 5000 + 100 = 5100 seeks
  - with *takes* as the outer relation
    - 10000 * 100 + 400 = 1,000,400 block transfers and 10,400 seeks
- If a relation (e.g. *student*) fits entirely in memory, the cost estimate will be 500 block transfers.
- Block nested-loops algorithm (next slide) is preferable.

Block Nested-Loop Join

- Variant of nested-loop join in which every block of inner relation is paired with every block of outer relation.

  for each block B_r **of** *r* **do begin**
      for each block B_s **of** ***s*** **do begin**
          for each tuple t_r **in** B_r **do begin**
              for each tuple t_s **in** B_s **do begin**
                  Check if (t_r, t_s) satisfy the join condition
                  if they do, add $t_r \bullet t_s$ to the result.
              end
          end
      end
  end

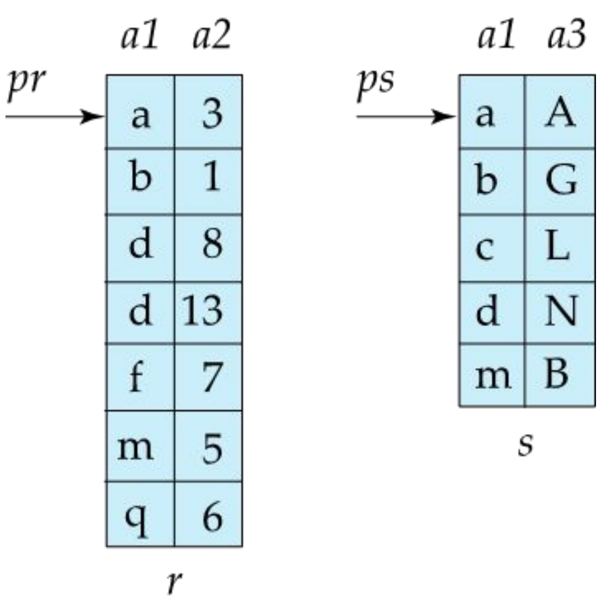
- Worst case estimate (neither relation fits in memory): $b_r * b_s + b_r$ block transfers and $2 * b_r$ seeks
  - Each block in the inner relation *s* is read once for each *block* in the outer relation
- Best case (inner relation fits in memory): $b_r + b_s$ block transfers and 2 seeks.
- A few more improvements are discussed in book
  - One of them is using index on inner relation if available (next slide)

Indexed Nested-Loop Join

- Index lookups can replace file scans if
  - join is an equi-join or natural join and
  - an index is available on the inner relation's join attribute
    - Can construct an index just to compute a join.
- For each tuple t_r in the outer relation *r,* use the index to look up tuples in *s* that satisfy the join condition with tuple t_r.
- Worst case: buffer has space for only one page of *r*, and, for each tuple in *r*, we perform an index lookup on *s.*
- Cost of the join: $b_r (t_T + t_S) + n_r * c$
  - Where *c* is the cost of traversing index and fetching all matching *s* tuples for one tuple of *r*
  - *c* can be estimated as cost of a single selection on *s* using the join condition.
- If indices are available on join attributes of both *r* and *s,* use the relation with fewer tuples as the outer relation.

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
   a. Join step is similar to the merge stage of the sort-merge algorithm.
   b. Main difference is handling of duplicate values in join attribute — every pair with same value on join attribute must be matched
   c. Merge-Join Algorithm Summary (detailed algorithm in book):



      1. Keep moving *ps* down, collect all tuples from *s* that have the same join attribute value and put them into a set, S_s
      2. Keep moving *pr* down, while the *pr* currently points to a tuple in *r* that has the value for join attribute which is less than those of tuples in S_s
      3. For each tuple in *r* that has the join attribute value, perform concatenation of it with each tuple in S_s
      4. repeat from 1 until *ps* and *pr* become null

Merge-Join (Cont.)

- Can be used only for equi-joins and natural joins
- Each block needs to be read only once (assuming all tuples for any given value of the join attributes fit in memory)
- Thus, the cost of merge join is:

  $b_r + b_s$ block transfers and $\lceil b_r / b_b \rceil + \lceil b_s / b_b \rceil$ seeks and the cost of sorting if relations are unsorted.

  b_b: Number of buffer blocks allocated to each relation

- **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 is more efficient than random lookup

Overview

- Join Operation
- Assignments

Assignments

- Reading: Ch15.5.1, 15.5.2, 15.5.3, 15.5.4
- Practice Excercises: n/a

Solutions to the Practice Excercises:
https://www.db-book.com/Practice-Exercises/index-solu.html

The End