Handling Other Operations

Chapter 12 and 14

Operator Evaluation

- How to implement common operators?
  - Selection
  - Join
  - Projection (optional DISTINCT)
  - Set Difference
  - Union
  - Aggregate operators (SUM, MIN, MAX, AVG)
  - GROUP BY

Projection

- `Select R.a, R.d FROM …`
  - Straightforward implementation!
- Combine with previous operation
- Zero Cost!!!

Question?

```
Select R.a, R.d
```
retain duplicate

Is exactly the SQL equivalent of the RA expression:

$\pi_{a,d}$ R

remove duplicate

A. True
B. False ✓

difference: dupliate elimination

Projection

- `Select DISTINCT R.a, R.d`
  - Remove attributes
  - Eliminate duplicates (not free)
- Algorithms for Projection `DISTINCT`:
  - 1 Sorting: Sort on <u>all</u> the projected attributes
    - Pass 0: eliminate unwanted fields. Tuples in the sorted-runs may be smaller
    - Eliminate duplicates in the merge pass & sort

      eliminate duplicate while sorting
  - 2 Hashing: Two phases
    - Partitioning
    - Duplicate elimination

Sort- or Hash-based: Which one?

- Sort-based approach
  - better handling of skew a lot of duplicates
  - result is sorted
  - Thus, more commonly used than hash-based approach

Projection – Index-Only Scans

- **Index-only scan**
  - Projection attributes subset of index attributes
  - Apply projection techniques (e.g., sorting and removing duplicates) to data entries (much smaller!)

- Special case of Index-only scan: if an ordered (i.e. tree) index contains all projection attributes as *prefix* of search key:
  - Retrieve index data entries in order (no sorting necessary)
  - Discard unwanted fields
  - Compare adjacent entries to eliminate duplicates (if required)

(a, b, c)

Operator Evaluation

- How to implement common operators?
  - Selection
  - Projection (optional DISTINCT)
  - Join
  - Set Difference
  - Union
  - Aggregate operators (SUM, MIN, MAX, AVG)
  - GROUP BY

Set Operations

- ∪ and – similar; we will do ∪
  - Both require duplicate elimination
- Duplicate elimination algorithms for ∪:
  1. Sorting:
     - Sort both relations (on all attributes). both table
     - Merge sorted relations eliminating duplicates.
  2. Hashing:
     - Partition R and S
     - Build hash table for R_i.
     - Probe with tuples in S_i, add to table if $\boxed{\text{not a}}$ duplicate

grace hash join: in it ⇒ take that
here: not in ⇒ take that

☆ Costs are the same
☆ structured the same

Question?

Are ∩ and X special cases of join?

I.e., join algorithms can be used to implement them.

A. True, True ✓
B. True, False
C. False, True
D. False, False

Yes: for general join algorithm

join on all attributes: same logic as join.

Operator Evaluation

- How to implement common operators?
  - Selection
  - Projection (optional DISTINCT)
  - Join
  - Set Difference
  - Union
  - Aggregate operators (SUM, MIN, MAX, AVG)
  - GROUP BY

Aggregates

- Sorting Approach
  - Sort on `GROUP BY` attributes (if any)
  - Scan sorted tuples, computing running aggregate
    - Min, Max
    - Count
    - Sum
    - Average: compute from sum and count
  - During scan, when the group by attribute changes (e.g. A,A,A,**B**), output aggregate result

* one buffer page needed in memory + some variables for counter (another page)

costs one full scan

from A, A, A, A to A, A, A, B : finish a group

Buffer Space?

- Scanning table, 1 page.
- Running aggregate(s), 1 page.

How can you use more memory?

No need to sort!! Compute multiple group aggregates in parallel.

* counter for each group ⇒ increment counter while full scan.

⇒ if too many counters to fit in memory ⇒ do partial sorted ⇒ apply this trick.

Aggregates

- Hashing Approach
  - Hash on GROUP BY attributes (if any)

pass one:

    - Hash entry: grouped attributes ① + running aggregate ②
  - Scan tuples, probe hash table, update hash entry
  - Scan hash table, and output each hash entry
- Cost: Scan of the relation! + read hash table

Aggregates

- Using an Index on aggregated attributes
  - Without Grouping
    - Can use B+-tree to aggregate attribute(s)

walk through tree leaves (cheaper than walk through data)

  - With grouping

best case: b+ organize by [a, b, c], group by [a, b], aggregate attr: c

    - B+-tree on all attributes in SELECT, WHERE and GROUP BY clauses

b+ order [c, a, b]
group by [a, b]
aggregate attr.

      - Index-only scan
      - If group-by attributes prefix of search key => data entries/tuples retrieved in group-by order

⇒ full scan leaves (cheaper access)

      - Else => get data entries and then use a sort or hash aggregate algorithm

as long as group by & aggr attr. are part of key.

Summary

- Various algorithms to choose from for each operator:
  - Selection
  - Join
    - Simple / Page / Block Nested Loops
    - Merge-Join
    - Hash Join (to be continued)
  - Projection (optional DISTINCT)
  - Set Difference
  - Union
  - Aggregate operators (SUM, MIN, MAX, AVG)
  - GROUP BY

Optional Exercises

- 12.1 (1-4), 12.3, 12.5
- 13.1, 13.3
- 14.1 (2, 3, 4, 6, 7, 8, 9, 10), 14