Query Processing

1. Basic Steps of Query Porcessing

Parsing and translation → relational algebra form → Optimization →

1.1. Parsing and translation

- Translate your SQL code into relational algebra expression(internal form)
- Parser check syntax and verify relations

1.2. Optimization

- Selecting the most efficient *evaluation plan* for given query
  - **Evaluation plan** : Set of *primitive operations*
    - **primitive operations** : relational algebra + evaluate method
    - e.g. $\sigma_{balance > 25000}(account): use\ index\ 1$
- Generating evaluation plan : **cost-based optimization**
  1. Generating logically *equivalent expressions* using equivalence rules
     - **Equivalent expressions** : if two query Q, R answer same result on any instance of same schema
  2. Annotating resulting expressions to get alternative query plans
  3. Choosing the cheapest plan baed on estimated cost

1.3. Measures of Query Cost : especially for time cost!

- Main parameter used for cost measuring
  1. number of seeks
  2. number of block transfers

- **Not** include final output writing

2. Selection Operation

2.1. Linear Searching : Scanning all block

- Cost estimate
    - Normal case : $b_r \times t_T + 1 \times t_S$
    - Key attribute : $(b_r/ 2) \times t_T + 1 \times t_S$
    - Why seek time is 1?
        - All blocks all sequentially linked, so we don't have to seek next block

2.2. Primary Index Searching

1. Key searching

    - simply, just following b+-tree
    - Cost estimate : $h_i \times (t_T + t_S) + 1 \times (t_T + t_S)$
        - b+-tree searching + get data cost

2. non-Key searching

    - Following b+-tree, and linear sacnning leaf node
    - Cost estimate : $h_i \times (t_T + t_S) + b \times t_T + t_S$
        - b+-tree searching + data linear scanning

3. Comparision included selection

    1. Bigger (+ equal) case
        - Find first index that match with condition and linearly scanning
    2. Lower (+ equal) case
        - From first block to matching data : **doesn't need to use index**

2.3. Secondary Index Searching

1. Key searching

    - same as primary index case
    - Cost estimate : $h_i \times (t_T + t_S) + 1 \times (t_T + t_S)$

2. non-Key searching

    - We can't ensure data sorted sequentially → get each data from different block
    - Cost estimate : $h_i \times (t_T + t_S) + n \times (t_T + t_S)$
        - b+-tree searching + get n data from different n blocks
        - Some time, full-sacn is could be better...

3. Comparision included selection

    1. Bigger (+ equal) case
        - Use index to find first entry and linally scanning leaf node and data

        2. Lower (+ equal) case
            - From first leaf node to first matching node
    - In this case, normally linear scan could be better option

3. Sort Operation

3.1. External Sort-Merge : for M main memory block and N data block

1. Create sorted runs
    - Read M block from disk
    - Sorting each M in-main memory block
    - Write data to disk
2. Merge the runs(N-way merge)
    - Read M-1 block : left 1 for writing
    - iterate in-memory block and find porper data
        - delete that proper data from memory, if memory is empty, than read next sorted block
        - if output block full, then write to disk sequencially
    - iterate process until all block is sorted

- Cost Analysis
    - Block transfer : for each merge process, b_r(read) + b_r(write) transfer needed
        - So, we need $2b_r + 2b_r \times log_{M-1}(b_r/M)$ block transfer
    - Seeks : $2\lceil b_r / M \rceil + \lceil b_r / b_b \rceil(2\lceil log_{M-1}(b_r / M)\rceil - 1)$

4. Join Operation

4.1. Nested-Loop Join

- Simple case : just loop!
    - $n_r \times b_s + b_r$ block transfer
        - full scanning s relation for each elements in b relation
        - so, b_s for n_r and single b_r
    - $n_r + b_r$
        - n_r means single seek for all b relation's elements
        - b_r means all b relation's block seeking

4.2. Block Nested-Loop Join

- Iterate s relation for in-memory block of b relation

```
for b_r in B_r:
for b_s in B_s:
    for t_r in b_r:
    for t_s in b_s:
        if f(t_r, t_s):
            add t_r + t_s to result
```

- Cost estimate
    - Block transfer : $b_r + \lceil b_r / (M - 2) \rceil \times b_s$
        - 2 for output and s relation block so, we need full scan s relation for $\lceil b_r / (M - 2) \rceil$ times
    - Seeks : $\lceil b_r / (M - 2) \rceil + \lceil b_r / (M - 2) \rceil$
        - one for b, and one for s

4.3. Hash-Join

- Making parition of b, s relation by using hash function

    - then how many parition do we need?
        - best must be a M-1(1 for input)

- Cost estimate

    - Block transfer : $3 \times (b_r + b_s) + 4 \times n_p$
        - $b_r + b_s$ for read relation
        - $b_r + b_s + 2 \times n_p$ for write parted block
        - $b_r + b_s + 2 \times n_p$ for read for join
    - Seeks : $2 \times (\lceil b_r / b_b \rceil + \lceil b_r / b_b \rceil) + 2\times n_p$
        - seek for build : $\lceil b_r / b_b \rceil + \lceil b_r / b_b \rceil$
        - seek for join : $\lceil b_r / b_b \rceil + \lceil b_r / b_b \rceil + 2\times n_p$