COL 362 & COL 632

Return of Query Processing

15 Mar 2023

Recap

SQL Query

↓

Logical Query Plan

↓

Physical Query Plan

↓

Execute

- Utilize logical plan equivalences to rewrite queries to those which are more efficient
- Choose the most efficient physical operator
- Order the overall query plan to minimize the cost
- For simplicity we just use the number of block transfers *from disk and the* number of seeks as the cost measure
  - t_T – time to transfer one block
  - t_S – time for one seek
  - Cost for b block transfers plus S seeks: $b * t_T + S * t_S$
- We ignore CPU costs

Measure of Query Cost – in reality

- Required data may be buffer resident already, avoiding disk I/O
  - But hard to take into account for cost estimation
- Several algorithms can reduce disk IO by using extra buffer space
  - Amount of real memory available to buffer depends on other concurrent queries and OS processes, known only during execution
- **Worst case** estimates assume that no data is initially in buffer and only the minimum amount of memory needed for the operation is available
  - But more optimistic estimates are used in practice

Selection Operation

linear $= b_r$ read + seek

binary $= \log_2 b_r \times$ read + $\log_2 b_r \times$ seek

- **File scan**
- Algorithm **A1** (**linear search**). Scan each file block and test all records to see whether they satisfy the selection condition.
  - Cost estimate = b_r block transfers + 1 seek
    - b_r denotes number of blocks containing records from relation r
  - If selection is on a key attribute, can stop on finding record
    - cost = $(b_r/2)$ block transfers + 1 seek

- Can we use binary search?

sorted on the selection attribute

seek count goes up

Selections Using Indices

$\sigma_{k=v}$

- **Index scan** – search algorithms that use an index
  - selection condition must be on search-key of index.
- **A2** (**clustering index, equality on key**). Retrieve a single record that satisfies the corresponding equality condition
  - *Cost* = $(h_i + 1) * (t_T + t_S)$

B+ tree

Hash

- **A3** (**clustering index, equality on nonkey**) Retrieve multiple records.

No overflow buckets

  - Records will be on consecutive blocks
    Let b = number of blocks containing matching records
  - *Cost* = $h_i * (t_T + t_S) + t_S + t_T * b$

$(h_i + 1)(t_s + t_T) + (b-1) \times t_T$

Selections Using Indices

- **A4** (**secondary index, equality on key/non-key**).
  - Retrieve a single record if the search-key is a candidate key
    Cost = $(h_i + 1) * (t_T + t_S)$
  - Retrieve multiple records if search-key is not a candidate key
    => each of *n* matching records may be on a different block
    Cost = $(h_i + n) * (t_T + t_S)$
    - Can be very expensive!

Selections Involving Comparisons

- Can implement selections of the form $\sigma_{A \le V}(r)$ or $\sigma_{A \ge V}(r)$ by using
  - a linear file scan,
  - or by using indices in the following ways:
- **A5** (**clustering index, comparison**). (Relation is sorted on A)
  - For $\sigma_{A \ge V}(r)$ use index to find first tuple $\ge v$ and scan relation sequentially from there
  - For $\sigma_{A \le V}(r)$ just scan relation sequentially till first tuple > v; do not use index
- **A6** (**clustering index, comparison**).
  - For $\sigma_{A \ge V}(r)$ use index to find first index entry $\ge v$ and scan index sequentially from there, to find pointers to records.
  - For $\sigma_{A \le V}(r)$ just scan leaf pages of index finding pointers to records, till first entry > v.
  - In either case, retrieve records that are pointed to
  - requires an I/O per record; Linear file scan may be cheaper!

$\theta_1 \wedge \theta_2 \equiv \theta_2 \wedge \theta_1$

Implementation of Complex Selections

$\underbrace{A_1 > V_1}_{\theta_1} \wedge \underbrace{A_2 = V_2}_{\theta_2}$

- **Conjunction:** $\sigma_{\theta_1 \wedge \theta_2 \wedge \ldots \theta_n}(r)$
- **A7 and A8**
- **A9** (**conjunctive selection by intersection of identifiers**).
  - Requires indices with record pointers.
  - Use corresponding index for each condition, and take intersection of all the obtained sets of record pointers.
  - Then fetch records from file
  - If some conditions do not have appropriate indices, apply test in memory.

Algorithms for Complex Selections

HW

- **Disjunction:** $\sigma_{\theta 1 \vee \theta 2 \vee \dots \theta n}(r)$.
- **A10** (**disjunctive selection by union of identifiers**).
  - Applicable if *all* conditions have available indices.
    - Otherwise use linear scan.
  - Use corresponding index for each condition and take union of all the obtained sets of record pointers.
  - Then fetch records from file
- **Negation:** $\sigma_{\neg\theta}(r)$
  - Use linear scan on file
  - If very few records satisfy $\neg\theta$, and an index is applicable to θ
    - Find satisfying records using index and fetch from file