Chapter 15: Query Processing

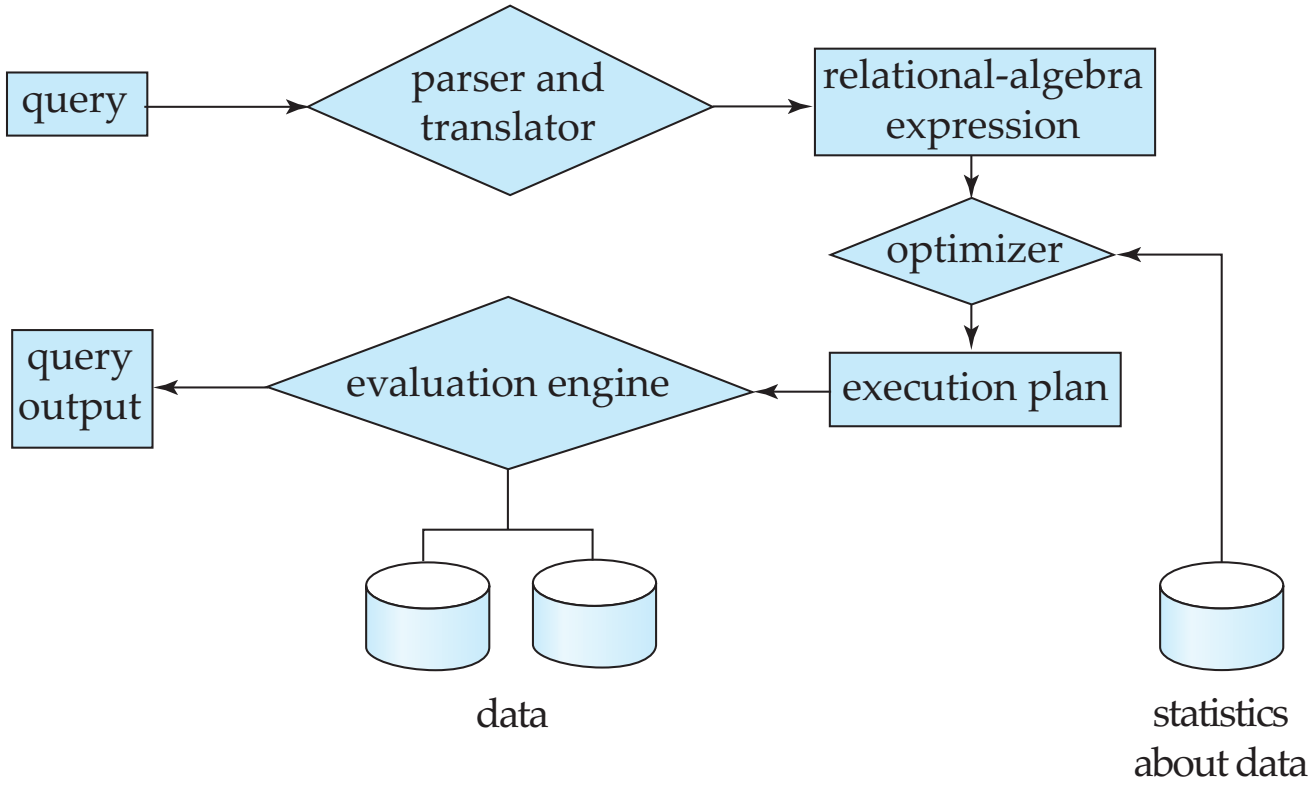
Database System Concepts, 7th Ed.

Chapter 15: Query Processing

- Overview
- Measures of Query Cost
- Selection Operation
- Sorting
- Join Operation
- Other Operations
- Evaluation of Expressions

Basic Steps in Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation

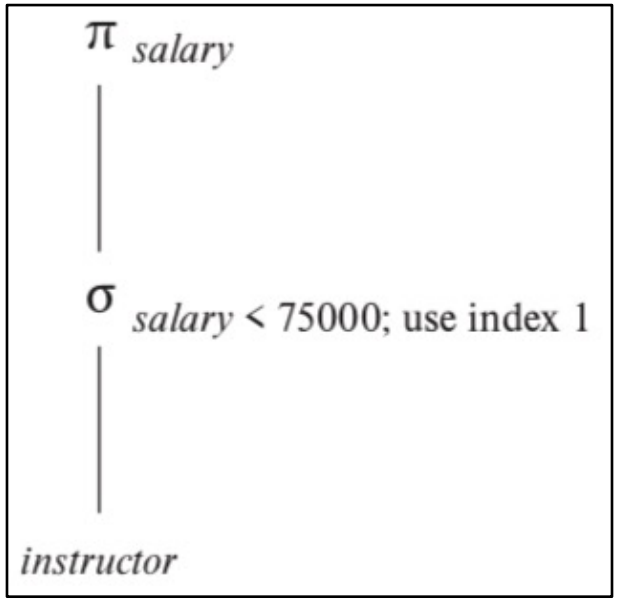


Basic Steps in Query Processing (Cont.)

- Parsing and translation
  - Translate the query into its internal form. This is then translated into relational algebra.
  - Parser checks syntax, verifies relations
- Evaluation
  - The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query.

Basic Steps in Query Processing: Optimization

- A relational algebra expression may have many equivalent expressions
  - E.g., $\sigma_{salary<75000}(\Pi_{salary}(instructor))$ is equivalent to $\Pi_{salary}(\sigma_{salary<75000}(instructor))$
- Each relational algebra operation can be evaluated using one of several different algorithms
  - Correspondingly, a relational-algebra expression can be evaluated in many ways.
- Annotated expression specifying detailed evaluation strategy is called an **evaluation-plan**. E.g.,:
  - Use an index on *salary* to find instructors with salary < 75000,
  - Or perform complete relation scan and discard instructors with salary ≥ 75000



Basic Steps: Optimization (Cont.)

- **Query Optimization**: Amongst all equivalent evaluation plans choose the one with lowest cost.
  - Cost is estimated using statistical information from the database catalog
    - e.g.. number of tuples in each relation, size of tuples, etc.

- In this chapter we study
  - How to measure query costs
  - Algorithms for evaluating relational algebra operations
  - How to combine algorithms for individual operations in order to evaluate a complete expression

Selection Operation

- **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
  - Linear search can be applied regardless of
    - selection condition or
    - ordering of records in the file, or
    - availability of indices
- Note: binary search generally does not make sense since data is not stored consecutively
  - except when there is an index available,
  - and binary search requires more seeks than index search

Selections Using Indices

- **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
  - $Cost = (h_i + 1) * (t_T + t_S)$

- **A3** (**clustering index, equality on nonkey**) Retrieve multiple records.
  - Records will be on consecutive blocks
    - Let b = number of blocks containing matching records
    - The leaf blocks assumed to be stored sequentially
  - $Cost = h_i * (t_T + t_S) + t_S + t_T * b$

Selections Using Indices

- **A4** (**secondary index, equality on key/non-key**).
  - Retrieve a single record if the search-key is a candidate key
    - Same as A3
    - *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
      - A seek per record may be required
    - 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** (**secondary 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!

Implementation of Complex Selections

- **Conjunction:** $\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots \theta n}(r)$

- **A7** (**conjunctive selection using one index**).
  - Select a combination of θ_i and algorithms A1 through A6 that results in the least cost for $\sigma_{\theta i}(r)$.
  - The cost of algorithm A7 is given by the cost of the chosen algorithm.

- **A8** (**conjunctive selection using composite index**).
  - Use appropriate composite (multiple-key) index if available.

- **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

- **Disjunction:** $\sigma_{\theta 1 \vee \theta 2 \vee \ldots \theta n}(r)$.

- **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