Module 56

Partha Pratim Das

Week Recap

Objectives & Outline

Query Processing

Query Cost

Selection Operation

Complex Selections

Sorting

External Sort-Merge

Join Operation

Other Operations

Module Summary

Database Management Systems

Module 56: Query Processing and Optimization/1: Processing

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Week Recap

Module 56

Partha Pratim Das

Week Recap

Objectives & Outline

Query Processing

Query Cost

Selection Operation

Complex Selections

Sorting

External Sort-Merge

Join Operation

Other Operations

Module Summary

- Learnt the importance of backup an analysed different backup strategies
- Failures may be due to variety of sources – each needs a strategy for handling
- A proper mix and management of volatile, non-volatile and stable storage can guarantee recovery from failures and ensure Atomicity, Consistency and Durability
- Log-based recovery is efficient and effective
- Learnt how Hot backup of transaction log helps in recovering consistent database.
- Studied the recovery algorithms for concurrent transactions
- Recovery based on operation logging supplements log-based recovery
- Planning for Backup
- Understood RAID - array of redundant disks in parallel to enhance speed and reliability

Module Objectives

Module 56

Partha Pratim Das

Week Recap

Objectives & Outline

Query Processing

Query Cost

Selection Operation

Complex Selections

Sorting

External Sort-Merge

Join Operation

Other Operations

Module Summary

- To understand the overall flow for Query Processing
- To define the Measures of Query Cost
- To understand the algorithms for processing Selection Operations, Sorting, Join Operations, and a few Other Operations

Module Outline

Module 56

Partha Pratim Das

Week Recap

Objectives & Outline

Query Processing

Query Cost

Selection Operation

Complex Selections

Sorting

External Sort-Merge

Join Operation

Other Operations

Module Summary

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

Module 56

Partha Pratim Das

Week Recap

Objectives & Outline

Query Processing

Query Cost

Selection Operation

Complex Selections

Sorting

External Sort-Merge

Join Operation

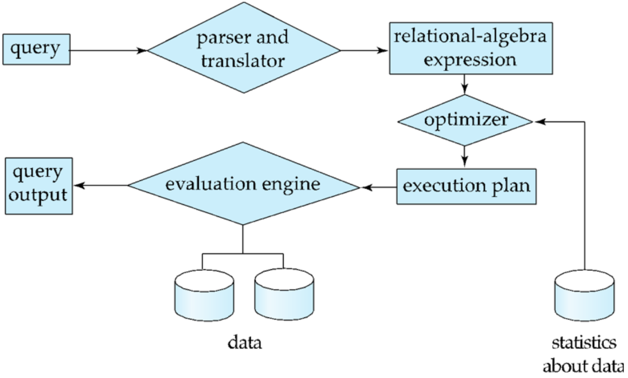
Other Operations

Module Summary

Overview of Query Processing

Basic Steps in Query Processing

a) Parsing and translation
b) Optimization
c) Evaluation



Basic Steps in Query Processing (2)

Module 56
Partha Pratim Das
Week Recap
Objectives & Outline
Query Processing
Query Cost
Selection Operation
Complex Selections
Sorting
External Sort-Merge
Join Operation
Other Operations
Module Summary

- 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 (3): Optimization

- Consider the query

  select salary
  from instructor
  where salary < 75000;

  which can be translated into either of the following relational-algebra expressions:
  - $\sigma_{salary<75000}(\Pi_{salary}(instructor))$
  - $\Pi_{salary}(\sigma_{salary<75000}(instructor))$
- 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**.
  - For example, can use an index on salary to find instructors with salary < 75000,
  - or can perform complete relation scan and discard instructors with salary ≥ 75000

Basic Steps in Query Processing (4): Optimization

Module 56

Partha Pratim Das

Week Recap

Objectives & Outline

Query Processing

Query Cost

Selection Operation

Complex Selections

Sorting

External Sort-Merge

Join Operation

Other Operations

Module Summary

- **Query Optimization**: Amongst all equivalent evaluation plans choose the one with lowest cost
  - Cost is estimated using statistical information from the database catalog
    - ▷ For example, number of tuples in each relation, size of tuples, etc.
- In this module 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
- In the next module
  - We study how to optimize queries, that is, how to find an evaluation plan with lowest estimated cost

Module 56

Partha Pratim Das

Week Recap

Objectives & Outline

Query Processing

Query Cost

Selection Operation

Complex Selections

Sorting

External Sort-Merge

Join Operation

Other Operations

Module Summary

Measures of Query Cost

Measures of Query Cost

- Cost is generally measured as total elapsed time for answering query
  - Many factors contribute to time cost
    - *disk accesses*, *CPU*, or even network *communication*
- Typically disk access is the predominant cost, and is also relatively easy to estimate
- Measured by taking into account
  - Number of seeks * average-seek-cost
  - Number of blocks read * average-block-read-cost
  - Number of blocks written * average-block-write-cost
    - Cost to write a block is greater than cost to read a block
      - data is read back after being written to ensure that the write was successful

Measures of Query Cost (2)

- For simplicity we just use the **number of block transfers** *from disk and the* **number of seeks** as the cost measures
  - 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 for simplicity
  - Real systems do take CPU cost into account
- We do not include cost to writing output to disk in our cost formulae

Measures of Query Cost (3)

Module 56

Partha Pratim Das

Week Recap

Objectives & Outline

Query Processing

Query Cost

Selection Operation

Complex Selections

Sorting

External Sort-Merge

Join Operation

Other Operations

Module Summary

- 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
    - We often use worst case estimates, assuming only the minimum amount of memory needed for the operation is available
- Required data may be buffer resident already, avoiding disk I/O
  - But hard to take into account for cost estimation

Module 56

Partha Pratim Das

Week Recap

Objectives & Outline

Query Processing

Query Cost

Selection Operation

Complex Selections

Sorting

External Sort-Merge

Join Operation

Other Operations

Module Summary

Selection Operation

Selection Operation: File / Index Scan

A#	Algorithm	Cost	Reason
A1	Linear Search	$t_S + b_r \times t_T$	One initial seek plus b_r block transfers
A1	Linear Search, Eq. on Key	Average case $t_S + (b_r/2) \times t_T$	Since at most one record satisfies condition, scan can be terminated as soon as the required record is found. b_r blocks transfers in worst case
A2	Prm. Index, Eq. on Key	$(h_i+1)\times(t_T+t_S)$	Index lookup traverses the height of the tree plus one I/O to fetch the record; each of these I/O operations requires a seek and a block transfer
A3	Prm. Index, Eq. on Nonkey	$h_i \times (t_T + t_S)+$ $b \times t_T$	One seek for each level of the tree, one seek for the first block. Here all of b are read. These blocks are leaf blocks assumed to be stored sequentially (for a primary index) and don't require additional seeks
A4	Snd. Index, Eq. on Key	$(h_i+1)\times(t_T+t_S)$	This case is similar to primary index
A4	Snd. Index, Eq. on Nonkey	$(h_i+n)\times(t_T+t_S)$	Here, cost of index traversal is the same as for $A3$, but each record may be on a different block, requiring a seek per record. Cost is potentially very high if n is large
A5	Prm. Index, Comparison	$h_i \times (t_T + t_S)+$ $b \times t_T$	Identical to the case of $A3$, equality on nonkey
A6	Snd. Index, Comparison	$(h_i+n)\times(t_T+t_S)$	Identical to the case of $A4$, equality on nonkey

t_T is time to transfer one block. t_S is time for one seek
b_r denotes the number of blocks in the file
b denotes the number of blocks containing records with the specified search key
h_i denotes the height of the index. n is the number of records fetched

Complex Selections: Conjunction

- **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)
  - Test other conditions on tuple after fetching it into memory buffer
- **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

Complex Selections: Disjunction

- **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

Module 56

Partha Pratim Das

Week Recap

Objectives & Outline

Query Processing

Query Cost

Selection Operation

Complex Selections

Sorting

External Sort-Merge

Join Operation

Other Operations

Module Summary

Sorting

Sorting

Module 56

Partha Pratim Das

Week Recap

Objectives & Outline

Query Processing

Query Cost

Selection Operation

Complex Selections

Sorting

External Sort-Merge

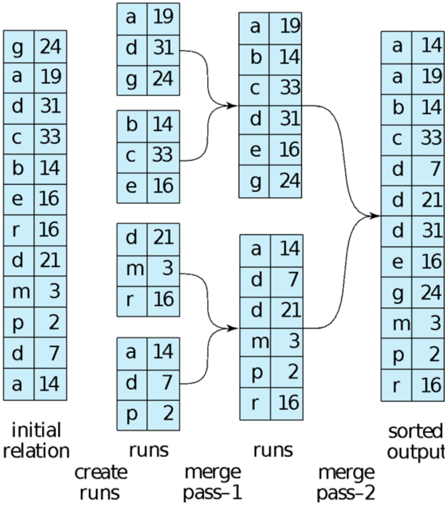
Join Operation

Other Operations

Module Summary

- We may **build an index on the relation**, and then use the index to read the relation in sorted order
  - May lead to one disk block access for each tuple
- For relations that fit in memory, techniques like **quicksort** can be used
- For relations that do not fit in memory, **external sort-merge** is a good choice

External Sort-Merge: Example



External Sort-Merge: Algorithm

a) **Create sorted runs**. Let `M` denote the number of blocks in the main-memory buffer available for sorting. First, a number of sorted runs are created; each run is sorted, but contains only some of the records of the relation.

```
i = 0;
repeat
    read M blocks of the relation, or the rest of the relation, whichever is smaller;
    sort the in-memory part of the relation;
    write the sorted data to run file Ri;
    i = i + 1;
until the end of the relation
```

b) **Merge the runs (N-way merge)**: Now, the runs are merged. For the total number of runs, `N < M`, so that we can allocate one block to each run and have space left to hold one block of output. The merge stage operates as follows:

```
read one block of each of the N files Ri into a buffer block in memory;
repeat
    choose the first tuple (in sort order) among all buffer blocks;
    write the tuple to the output, and delete it from the buffer block;
    if the buffer block of any run Ri is empty and not end-of-file(Ri)
    then read the next block of Ri into the buffer block;
until all input buffer blocks are empty
```

c) If $N \geq M$, several merge *passes* are required

- In each pass, contiguous groups of $M-1$ runs are merged.
- A pass reduces the number of runs by a factor of $M-1$, and creates runs longer by the same factor
  - For M=11 and 90 runs, one pass reduces the number of runs to 9, each 10 times the size of the initial runs
- Repeated passes are performed till all runs have been merged into one

Module 56

Partha Pratim Das

Week Recap

Objectives & Outline

Query Processing

Query Cost

Selection Operation

Complex Selections

Sorting

External Sort-Merge

Join Operation

Other Operations

Module Summary

Join Operation

Join Operation

Module 56

Partha Pratim Das

Week Recap

Objectives & Outline

Query Processing

Query Cost

Selection Operation

Complex Selections

Sorting

External Sort-Merge

Join Operation

Other Operations

Module Summary

- 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
- Examples use the following information
  - Number of records of *student*: $n_{students} = 5{,}000$
  - Number of records of *takes*: $n_{takes} = 10{,}000$
  - Number of blocks of *student*: $b_{students} = 100$
  - Number of blocks of *takes*: $b_{takes} = 400$

Nested-Loop Join

Module 56
Partha Pratim Das
Week Recap
Objectives & Outline
Query Processing
Query Cost
Selection Operation
Complex Selections
Sorting
External Sort-Merge
Join Operation
Other Operations
Module Summary

- 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 $t_r \bullet t_s$ to the result.

      end

  end
- 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 (2)

- 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 $n_r + b_r$ seeks, where n_r (n_s) denotes the number of tuples in r (s) and b_r (b_s) denotes the number of blocks containing tuples of in r (s)
- If the smaller relation fits entirely in memory, use that as the inner relation.
  - Reduces cost to $b_r + b_s$ block transfers and 2 seeks
- Example of join of *students* and *takes*: $n_{students} = 5{,}000$, $n_{takes} = 10{,}000$, $b_{students} = 100$, $b_{takes} = 400$
- Assuming 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 smaller relation (*student*) fits entirely in memory, the cost estimate will be 500 block transfers
- Block nested-loops algorithm is preferable

Block Nested-Loop Join

Module 56

Partha Pratim Das

Week Recap

Objectives & Outline

Query Processing

Query Cost

Selection Operation

Complex Selections

Sorting

External Sort-Merge

Join Operation

Other Operations

Module Summary

- 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

Block Nested-Loop Join (2)

Module 56
Partha Pratim Das
Week Recap
Objectives & Outline
Query Processing
Query Cost
Selection Operation
Complex Selections
Sorting
External Sort-Merge
Join Operation
Other Operations
Module Summary

- Worst case estimate: $b_r * b_s + b_r$ block transfers + 2 * b_r seeks
  - Each block in the inner relation s is read once for each block in the outer relation
- Best case: $b_r + b_s$ block transfers + 2 seeks.
- Improvements to nested loop and block nested loop algorithms:
  - In block nested-loop, use $M - 2$ disk blocks as blocking unit for outer relations, where M = memory size in blocks; use remaining two blocks to buffer inner relation and output
    - ▷ $Cost = \lceil b_r/(M-2) \rceil * b_s + b_r$ block transfers $+2 * \lceil b_r/(M-2) \rceil$ seeks
  - If equi-join attribute forms a key or inner relation, stop inner loop on first match
  - Scan inner loop forward and backward alternately, to make use of the blocks remaining in buffer (with LRU replacement)
  - Use index on inner relation, if available

Indexed Nested-Loop Join

Module 56

Partha Pratim Das

Week Recap

Objectives & Outline

Query Processing

Query Cost

Selection Operation

Complex Selections

Sorting

External Sort-Merge

Join Operation

Other Operations

Module Summary

- 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: $b_r \times (t_T + t_S) + n_r * c$
  - Where c is the cost of traversing index and fetching all matching s tuples for one tuple or 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.

Example of Nested-Loop Join Costs

- Compute *student* $\bowtie$ *takes*, with *student* as the outer relation.
- Let *takes* have a primary B^+-tree index on the attribute *ID*, which contains 20 entries in each index node.
- Since *takes* has 10,000 tuples, the height of the tree is 4, and one more access is needed to find the actual data
- *student* has 5000 tuples
- Cost of block nested loops join
  - 400*100 + 100 = 40,100 block transfers + 2 * 100 = 200 seeks
    - ▷ assuming worst case memory
    - ▷ may be significantly less with more memory
- Cost of indexed nested loops join
  - 100 + 5000 * 5 = 25,100 block transfers and seeks.
  - CPU cost likely to be less than that for block nested loops join

Module 56

Partha Pratim Das

Week Recap

Objectives & Outline

Query Processing

Query Cost

Selection Operation

Complex Selections

Sorting

External Sort-Merge

Join Operation

Other Operations

Module Summary

Other Operations

Other Operations

Module 56

Partha Pratim Das

Week Recap

Objectives & Outline

Query Processing

Query Cost

Selection Operation

Complex Selections

Sorting

External Sort-Merge

Join Operation

Other Operations

Module Summary

- **Duplicate Elimination**
- **Projection**
- **Aggregation**
- Set Operations
- Outer Join

Other Operations: Duplicate Elimination & Projection

Module 56

Partha Pratim Das

Week Recap
Objectives & Outline
Query Processing
Query Cost
Selection Operation
Complex Selections
Sorting
External Sort-Merge
Join Operation
Other Operations
Module Summary

- **Duplicate Elimination** can be implemented via hashing or sorting
  - On sorting duplicates will come adjacent to each other, and all but one set of duplicates can be deleted
  - *Optimization*: duplicates can be deleted during run generation as well as at intermediate merge steps in external sort-merge
  - Hashing is similar – duplicates will come into the same bucket
- **Projection** :
  - perform projection on each tuple
  - followed by duplicate elimination

Other Operations: Aggregation

Module 56
Partha Pratim Das
Week Recap
Objectives & Outline
Query Processing
Query Cost
Selection Operation
Complex Selections
Sorting
External Sort-Merge
Join Operation
Other Operations
Module Summary

- **Aggregation** can be implemented in a manner similar to duplicate elimination
  - Sorting or hashing can be used to bring tuples in the same group together, and then the aggregate functions can be applied on each group
  - *Optimization*: combine tuples in the same group during run generation and intermediate merges, by computing partial aggregate values
    - For count, min, max, sum: keep aggregate values on tuples found so far in the group
      - When combining partial aggregate for count, add up the aggregates
    - For avg, keep sum and count, and divide sum by count at the end

Module Summary

Module 56
Partha Pratim Das
Week Recap
Objectives & Outline
Query Processing
Query Cost
Selection Operation
Complex Selections
Sorting
External Sort-Merge
Join Operation
Other Operations
Module Summary

- Understood the overall flow for Query Processing and defined the Measures of Query Cost
- Studied the algorithms for processing Selection Operations, Sorting, Join Operations and a few Other Operations

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with "PPD".