Database Systems

Algorithms for Relational Algebra Operators and Query Evaluation

Dr P Sreenivasa Kumar

Professor

CS&E Department

I I T Madras

Relational Query Evaluation

- Relational Algebra Operators
  - Select, Project, Join
  - Union, Intersect, Difference
- Grouping and aggregation
  - Sorting
- How to implement these?
- How do indexes help?
- Any other information is helpful?

Selection With Equality Conditions

- Single selection condition $X = c_1$
  - Index on X ?   Yes: use the index;   No: file scan
- Several conjunctive conditions
  - $X_1 = c_1$ and $X_2 = c_2$ and … and $X_k = c_k$
    - Index on any X_i ?
    - Yes: Get the records and check other conditions
    - No: File scan
- Several disjunctive conditions
  - Index on any *single* X_i - not helpful
    - Difficult compared to conjunctive case

Predicate Selectivity

- Selectivity s of a condition C -- $0 \leq s \leq 1$
  - (No. of records satisfying C) / (Total no. of records)
    - C_1: student.dept = "CSE" -- 450 / 8000 = 0.056
    - C_2: student.sex = "female" -- 1200 / 8000 = 0.15
    - C_3: student.rollNo = "CS10B032" -- 1/8000 = 0.000125
  - highly selective predicate - very ***low*** selectivity value
- Conjunction of conditions
  - Choose the one that is *most* selective
    - Get the records and check other conditions
  - Selectivity values (estimates): collect offline

Selectivity Estimation

- Maintained in the DB *catalog*
  - Used by the query optimizer
- Equality conditions involving a key attribute
  - Selectivity = 1/ (Total no. of records)
- Equality conditions involving a non-key attribute
  - Selectivity = 1/ (Distinct values of the attribute)
- Sometimes histograms are also maintained
  - Distinct value or value range -- # of records

Project Operation

- For every record in the operand
  - Access it, take the required attributes values
  - Construct the result record
- Duplicate Elimination
  - Costly
  - Sort or hash based methods are used
- File scan becomes essential
- Apply project after selection, if possible
  - To reduce the input to project

External Sorting

- Sorting a file
  - An often required operation
    - Duplicate elimination, Grouping of records, Join etc
- Merge-sort Principle is used
  - O($n$log$n$) worst-case complexity for n items
  - Two phases
    - **Sort phase** – repeat: read part of data, sort and write
      - Create many sorted files – called *runs*
    - **Merge phase** – repeat: merge *some* sorted files and write
      - Till only one sorted file is left

Algorithm – Sort Phase

- File: n blocks and Buffer memory: m blocks
- Sort Phase
  - Repeat the following $\lceil n/m \rceil$ times

    {read the next m blocks; sort in-memory;

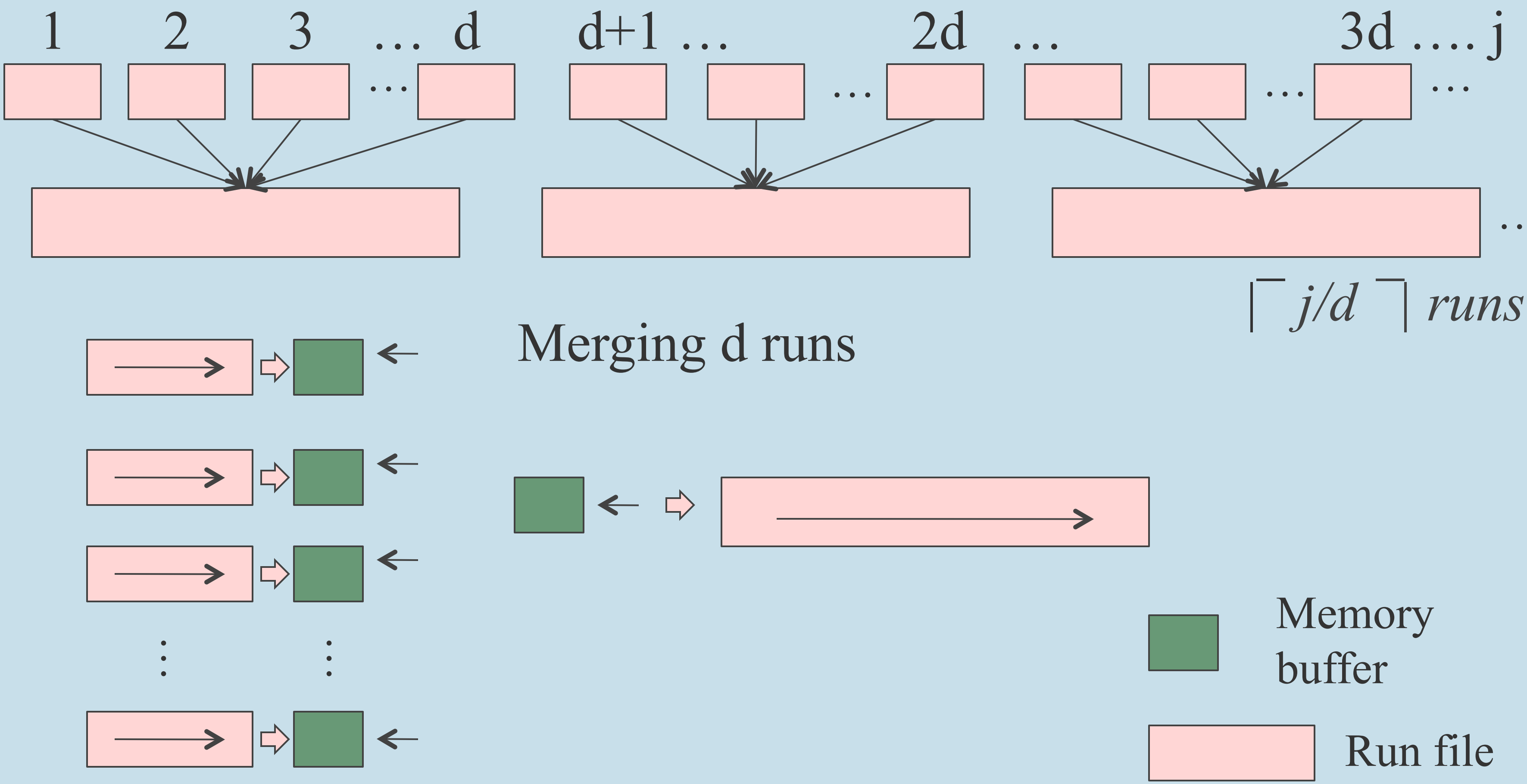
    write to disk as a single file, called a *run*}
- Number of *runs* $r = \lceil n/m \rceil$
- Complexity: n block reads and n block writes
  - $2n$ block accesses

Algorithm – Merge Phase

- File: n blocks, Memory Buffers: m (≥ 3) blocks, Runs: r
  - Degree of merging d : $2 \leq d \leq (m-1)$
- Merge Phase: repeat the following $\lceil log_d\, r \rceil$ times
  - Reduce j runs to $\lceil j/d \rceil$ runs (Initially, $j = r$)
    - By repeatedly merging d runs at a time to get *one* run
      - **Use *d* buffers, one for each of the next *d* runs; use one for the result**
      - **Get one block at a time from each run**
      - **Merge and write the result to disk – one block at a time**
- Complexity: $2n \lceil log_d\, r \rceil$
    - Each sub-phase : Entire file gets read and written
- Overall: $(2n + 2n \lceil log_d\, r \rceil)$ block accesses

Algorithm – Merge Phase

Reducing j runs to $\lceil j/d \rceil$ runs



Join Processing

- Join – A very important operation
- 2-way join
  - Two files of records, join condition – given
- Multi-way join
- Choice of algorithm depends on …
  - Sizes of files
  - Primary organization of the files
  - Availability of indices
  - Selectivity of the join condition etc

Nested Loop Join (or block nested loop join)

```
for each record x in R do
    for each record y in S do
        check if x, y join .. .
```

- Brute force join
- Two data files
  - R : b_1 blocks, S : b_2 blocks, Buffer : *m* blocks
- Buffer Usage: One block for the result of join
  - One for inner file (say, S); (m – 2) for outer file (R)
- For each *set* of (m – 2) blocks of R read-in, do
  - For each block of S do

    Read it in, compute join, write to result block

    Write the result block to disk whenever it fills up

Nested Loop Join - Time taken

- Two data files
  - R : b_1 blocks, S : b_2 blocks, Buffer : m blocks
- Outer file : b_1 blocks accesses
- # times inner file blocks accessed: $\lceil b_1/(m-2) \rceil$
- Overall: $b_1 + \lceil b_1/(m-2) \rceil\, b_2$
- Or, symmetrically: $b_2 + \lceil b_2/(m-2) \rceil\, b_1$
  - when we have S in the outer loop and R inside
- Which file in the outer loop?
  - The one with fewer blocks!

Nested Loop Join - Example

- Two data files

  R : b_1 = 5600 blocks, S : b_2 = 120 blocks, Buffer : 52 blocks

- If R is used in the outer loop
  - $b_1 + \lceil b_1/(m-2) \rceil b_2$
  - $5600 + \lceil 5600 / 50 \rceil * 120 = 19040$ disk ops
- If S is used in the outer loop
  - $120 + \lceil 120/ 50 \rceil * 5600 = 16920$ disk ops
- Assuming 10 msec per disk op
  - It is 190 secs versus 169 secs

Single Loop Join (or index loop join)

- Two data files
  - R : b_1 blocks, S : b_2 blocks
    - Need to compute **equi-join** with R.A = S.B
  - We have index on one of them, say S on B
- For each record *x* of R read in, do
  - Use the index on B for S
  - get all the matching records (having B = *x*.A)
- Time taken: $b_1 * h_B(S)$
  - $h_B(S)$ – # of block accesses of the index on B for S

Join Selection Factor

- Fraction of records in a file that join with records of the other for the given condition
- Consider: professor $\bowtie_{\text{empId = hod}}$ department
  - Only 5% of professor rows join with department rows
  - 100% of department rows join with professor rows
- Impacts performance of single loop join
  - If indexes are available on both files
  - Loop over records of the file with high join selection factor

Join Selection Factor - Example

- Impacts single loop join performance
  - If indexes are available on both files
- Consider: professor $\bowtie_{\text{empId = hod}}$ department
  - Loop over *professor* records and probe *department* using index on *hod* (option 1) OR
  - Loop over *department* records and probe *professor* using index on *empId* (option 2)
  - Option 1: 95% probes don't give a match
  - Option 2: All probes give a match
- Option 2 is the right choice

Hash Join

- Consider a 2-way equi-join $R \bowtie_{R.A=S.B} S$
  - Assume that S **fits into memory**
- Use a hash function *h*
  - Hash the records of S into M buckets using B-values
    - Called the **partitioning** of S
- To compute join result
  - Hash records of R, one by one, using A values
    - Use the ***same*** M buckets and the ***same*** hash function *h*
    - Matching pair of records will hash to same bucket

Partition Hash Join

- Consider a 2-way equi-join $R \bowtie_{R.A=S.B} S$
  - **Neither R nor S fits into the memory**
- Partition Phase: use a hash function *h*
  - Hash the records of R into *m* buckets using A-values
    - We get R_1 , R_2 , …, R_m - write them to files
  - Hash the records of S into *m* buckets using B-values
    - We get S_1 , S_2 , …, S_m - write them to files
  - Goals: ensure that distribution is uniform and
    - At least one of R_i or S_i fit into the memory
- To compute join result: join R_i with S_i only!

Partition Hash Join – Probe Phase

- Probe Phase: Join R_i with S_i for all i
- If one of R_i or S_i fit into the memory
  - Use the idea of hash join again!
    - Hash the smaller of the two into main memory using a *different* hash function, say h_2
    - Read the other file, probe and produce result records
  - Overall cost: (3(|R|+|S|) + |result|) block accesses
- Else use nested loops join
  - Overall cost: 2(|R|+|S|) + cost of nested loop joins

Sort-merge join

- Consider a 2-way equi-join $R \bowtie_{R.A=S.B} S$
- If R is sorted on A, S is sorted on B
  - Merge R and S to get join results
  - Called merge join - - very efficient - - linear
- If one of them is sorted on join attribute
  - Sorting the other and merging may be cost-effective
- Of course, we can
  - Sort R on A, sort S on B and use merge
  - Cost might be high

Set Operations

- Hash based join method
  - Can be adapted to compute Union, Intersect and Difference
- Sort-Merge method
  - Can be adapted to compute Union, Intersect and Difference
- Please study the details!

Query Optimization

- An SQL query - converted to a RA expression tree
- Initial RA expression is re-written
  - Using heuristic and algebraic transformation rules
    - Called algebraic optimization
  - Final RA expression tree is generated
- Cost-based query optimization
  - Cost estimates of *methods* for RA ops are computed
  - Execution plan with least estimated cost is chosen

Heuristic Optimization

- An SQL query - converted to a RA expression tree
- Initial RA expression is transformed
- Main heuristic rule
  - Apply *select* and *project* before other operations
    - Reduces the size of intermediate results
    - Reduces the number of fields in the intermediate results
- Make use of relational algebraic laws
  - *Select, project, join, union, intersect* - commutative
  - *Join, union, intersect* - associative
  - There are many more….(Read about them)

Cost-based Optimization

- Initial expression tree is rebuilt using heuristics and algebraic laws
- Each operator
  - Can be carried out using *many* methods
  - For a method, its cost function gives *estimated cost*
    - By taking file sizes, access path costs etc into account
  - Choice made at a node may effect choices at others
- Evaluate different plans based on estimated costs
  - Choose the plan with least cost

Query Optimization – Example

- Obtain the name and phone details of professors who taught the courses taken by student with roll number "CS08B027" in the even semester of 2010
- *select* p.empId, p.name, p.phone

  from professor p, teaching t, enrollment e

  where e.rollNo = "CS08B027" and e.courseId = t.courseId

  and e.sem = "even" and e.year = 2010

  and t.sem = "even" and t.year = 2010 and

  p.empId = t.empId

Query Optimization – Example

- Obtain the name and phone details of professors who taught the courses taken by student with roll number "CS08B027" in the even semester of 2010
- Initial RA Expr: $\Pi_{\text{p.empId, p.name, p.phone}} (\sigma_\theta (p \times t \times e))$ where

  p: professor, t: teaching, e: enrollment and

  θ = (e.rollNo = "CS08B027" and e.courseId = t.courseId
  and e.sem = "even" and e.year = 2010
  and t.sem = "even" and t.year = 2010 and
  p.empId = t.empId)

Query Optimization – Example

- $\Pi_{\text{p.empId, p.name, p.phone}} (\sigma_{\theta} (p \times t \times e))$

  $\equiv \Pi_{\text{p.empId, p.name, p.phone}} (\sigma_{\theta 3} (p \times \sigma_{\theta 2} (t) \times \sigma_{\theta 1}(e)) \ )$

  p: professor, t: teaching, e: enrollment and

  θ1 = (e.rollNo = "CS08B027" and

  and e.sem = "even" and e.year = 2010)

  θ2 = (t.sem="even" and t.year= 2010)

  θ3 = (p.empId=t.empId and e.courseId = t.courseId)

Query Optimization – Example

- $\Pi_{\text{p.empId, p.name, p.phone}} (\sigma_{\theta} (p \times t \times e))$

  $\equiv \Pi_{\text{p.empId, p.name, p.phone}} (\sigma_{\theta 3} (p \times \sigma_{\theta 2} (t) \times \sigma_{\theta 1}(e)))$

  $\equiv \Pi_{\text{p.empId, p.name, p.phone}} (\sigma_{\theta 3} (p \times \sigma_{\theta 4} (\sigma_{\theta 2} (t) \times \sigma_{\theta 1}(e))))$

  p: professor, t: teaching, e: enrollment and

  θ1 = (e.rollNo = "CS08B027" and

  e.sem = "even" and e.year = 2010)

  θ2 = (t.sem="even" and t.year= 2010)

  θ3 = (p.empId=t.empId)

  θ4 = (e.courseId = t.courseId)

Query Optimization – Example

- $\Pi_{\text{p.empId, p.name, p.phone}} (\sigma_\theta (p \times t \times e))$

  $\equiv \Pi_{\text{p.empId, p.name, p.phone}} (\sigma_{\theta 3} (p \times \sigma_{\theta 2} (t) \times \sigma_{\theta 1}(e)))$

  $\equiv \Pi_{\text{p.empId, p.name, p.phone}} (\sigma_{\theta 3} (p \times \sigma_{\theta 4} (\sigma_{\theta 2} (t) \times \sigma_{\theta 1}(e))))$

  $\equiv \Pi_{\text{p.empId, p.name, p.phone}} (p \bowtie_{\theta 3} (\sigma_{\theta 2}(t) \bowtie_{\theta 4} \sigma_{\theta 1}(e)))$

  θ1 = (e.rollNo = “CS08B027” and
  e.sem = “even” and e.year = 2010)

  θ2 = (t.sem=“even” and t.year= 2010)

  θ3 = (p.empId=t.empId)

  θ4 = (e.courseId = t.courseId)

Query Optimization – Example

- $\Pi_{\text{p.empId, p.name, p.phone}} (\sigma_{\theta} (p \times t \times e))$

  $\equiv \Pi_{\text{p.empId, p.name, p.phone}} (\sigma_{\theta 3} (p \times \sigma_{\theta 2} (t) \times \sigma_{\theta 1}(e)))$

  $\equiv \Pi_{\text{p.empId, p.name, p.phone}} (\sigma_{\theta 3} (p \times \sigma_{\theta 4} (\sigma_{\theta 2} (t) \times \sigma_{\theta 1}(e))))$

  $\equiv \Pi_{\text{p.empId, p.name, p.phone}} (p \bowtie_{\theta 3} (\sigma_{\theta 2}(t) \bowtie_{\theta 4} \sigma_{\theta 1}(e)))$

  $\equiv (\Pi_{\text{empId,name,phone}} (p) \bowtie_{\theta 3} \Pi_{\text{empId}} (\Pi_{\text{courseId, empId}} \sigma_{\theta 2} (t) \bowtie_{\theta 4} \Pi_{\text{courseId}} \sigma_{\theta 1} (e)))$

$\theta 1 =$ (e.rollNo = "CS08B027" and e.sem = "even" and e.year = 2010)

$\theta 2 =$ (t.sem="even" and t.year= 2010)

$\theta 3 =$ (p.empId = empId) $\theta 4 =$ (t.courseId = e.courseId)

Cost-based Optimization

- $\Pi_{\text{p.empId, p.name, p.phone}} (\sigma_{\theta} (p \times t \times e))$

  $\equiv \Pi_{\text{p.empId, p.name, p.phone}} (\sigma_{\theta 3} (p \times \sigma_{\theta 2} (t) \times \sigma_{\theta 1}(e)))$

  $\equiv \Pi_{\text{p.empId, p.name, p.phone}} (\sigma_{\theta 3} (p \times \sigma_{\theta 4} (\sigma_{\theta 2} (t) \times \sigma_{\theta 1}(e))))$

  $\equiv \Pi_{\text{p.empId, p.name, p.phone}} (p \bowtie_{\theta 3} (\sigma_{\theta 2} (t) \bowtie_{\theta 4} \sigma_{\theta 1} (e)))$

  $\equiv (\Pi_{\text{empId,name,phone}}(p) \bowtie_{\theta 3} \Pi_{\text{empId}} (\Pi_{\text{courseId, empId}} \sigma_{\theta 2} (t) \bowtie_{\theta 4} \Pi_{\text{courseId}} \sigma_{\theta 1} (e)))$

Evaluate costs of using different methods for the two selections, two joins and choose the least cost plan