15-721 Database Management Systems

Query Optimization

Instructor: Anastassia Ailamaki
http://www.cs.cmu.edu/~natassa

Roadmap - detailed

- Processing steps - overview
- Single-table query optimization
- Join query optimization
- Nested queries

Query Processing Phases

- Parsing
- Optimization
- Code Generation
- Execution
Types of queries
(query block)
- Single-table
- 2-way join
- n-way join
- nested subqueries

Access Paths
- Segment (Relation) Scan - each page is accessed exactly once
- Index Scan (B+ Tree)
  - Clustered
    - each index & data page: is touched once
  - Unclustered
    - each index page: touched once
    - each tuple may be touched once, but each page may be fetched multiple times

Join Methods
- Nested Loops
- Sort-merge
- (Hash join)
- Access path is orthogonal choice
Useful Definitions

- A SARGable predicate: attribute op value
- A SARG (Search ARGument for scans): a boolean expression of the SARGable predicates in disjunctive normal form: SARG1 or SARG2 or … or SARGn
  (SARG1 and … and SARGn) or
  (SARGn+1 and … and SARGq) or …

Definitions (cont.)

- A predicate (or set of predicates) matches an index when
  predicates are SARGable, and
  columns in the predicate are initial substring of index key

Example

- Index: name, location
  Predicates:
  "name = smith" matches index
  "name = smith or name = jones" matches
  "name = smith and location = San Jose" matches
  "(name = x and location = z) or (name = y and location = q)" matches
Definitions (cont.)

- An ordering of tuples is **interesting** if it is an ordering needed for a:
  - GroupBy,
  - OrderBy, or
  - Join

Roadmap - detailed

- Processing steps - overview
  - Single-table query optimization
  - Join query optimization
  - Nested queries

Single-Relation: Cost Model

- Cost of a Query - how would you measure it?
Single-Relation: Cost Model

- Cost of a Query =
  
  \[
  \# \text{page fetches} + W \times \# \text{RSI Calls}
  \]

- \((\# \text{RSI Calls}) = \# \text{tuples returned by RSI}\)

- \(W\) is a weighting factor
  
  - pages fetched vs. instructions executed

---

Single relation

- How to estimate \#I/O’s, for, say
  
  ```sql
  select *
  from EMP
  where salary > 30,000
  ```

---

Statistics

- What statistics would you need?
Statistics for Optimization

- NCARD(T) - cardinality of relation T in tuples
- TCARD(T) - number of pages containing tuples from T

Stats for Optimization (cont’d)

- P(T) = TCARD(T)/(# of non-empty pages in the segment)
  - If segments only held tuples from one relation there would be no need for P(T)
- ICARD(I) - number of distinct keys in index I
- NINDX(I) - number of pages in index I

Comments

- How / how-often would you update the stats?
### Comments
- Statistics not updated with each insert/delete/modify statement
- Generated at load time
- Update periodically using the `update statistics` command

### Single relation
- How to estimate #I/O’s, for, say `select *` from EMP where salary > 30,000

### Step #1 of Query Optimization
- Calculate a selectivity factor ‘F’ for each boolean factor in the predicate list
- Single-relation access paths
  - `a1 = value; a1=a2; value1<=a1<=value2`
  - `p or q; not p; p and q`
Predicate Selectivity Estimation

<table>
<thead>
<tr>
<th>attr = value</th>
<th>$F = \frac{1}{\text{ICARD}(\text{attr index})}$ – if index exists $F = \frac{1}{10}$ otherwise</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr1 = attr2</td>
<td></td>
</tr>
<tr>
<td>val1 &lt; attr &lt; val2</td>
<td></td>
</tr>
<tr>
<td>expr1 or expr2</td>
<td></td>
</tr>
<tr>
<td>expr1 and expr2</td>
<td></td>
</tr>
<tr>
<td>NOT expr</td>
<td></td>
</tr>
</tbody>
</table>

$F = \frac{1}{\text{max}(\text{ICARD}(I1),\text{ICARD}(I2))}$ or $F = \frac{1}{\text{ICARD}(Ii)}$ – if only index $i$ exists, or $F = \frac{1}{10}$
### Predicate Selectivity Estimation

<table>
<thead>
<tr>
<th>Condition</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr = value</td>
<td>$F = \frac{1}{ICARD(\text{attr index})}$ if index exists, otherwise $F = \frac{1}{10}$</td>
</tr>
<tr>
<td>attr1 = attr2</td>
<td>$F = \frac{1}{\max(ICARD(I1),ICARD(I2))}$ or $F = \frac{1}{ICARD(I)}$ if only index i exists, otherwise $F = 0.10$</td>
</tr>
<tr>
<td>val1 &lt; attr &lt; val2</td>
<td>$F = \frac{\text{value2-value1}}{\text{high key-low key}}$ if $\text{val1} &lt; \text{attr} &lt; \text{val2}$ else $F = 0.10$</td>
</tr>
<tr>
<td>expr1 or expr2</td>
<td>$F = \frac{1}{ICARD(I)}$ if only index i exists, otherwise $F = 0.10$</td>
</tr>
<tr>
<td>expr1 and expr2</td>
<td>$F = 1 - F(\text{expr1})F(\text{expr2})$</td>
</tr>
<tr>
<td>NOT expr</td>
<td>$F = F(\text{expr})$</td>
</tr>
</tbody>
</table>
Comments

- Query cardinality is the product of the relation cardinalities times the selectivities of the query’s boolean factor
  \[ \text{QCARD} = |R_1| \times |R_2| \times \ldots \times |R_n| \times F_{R_1} \times F_{R_2} \times \ldots \times F_{R_n} \]
- RSICARD (# RSI calls performed) = |R_1| \times |R_2| \times \ldots \times |R_n| \times \text{selectivity factors of all SARGABLE boolean factors}

Step #2 of Query Optimization

- For each relation, calculate the cost of scanning the relation for each suitable index, and a segment scan
- What is produced:
  i) Cost \( C \) in the form of # pages fetched + \( W \times \text{RSICARD} \)
  ii) Ordering of tuples the access path will produce

Costs per Access Path Case

<table>
<thead>
<tr>
<th>Case</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique index matching equal predicate</td>
<td>( 1+1+W )</td>
</tr>
<tr>
<td>Clustered index I matching ( \geq 1 ) preds</td>
<td>( F(\text{preds}) \times (\text{NINDX(I)}+\text{TCARD})+W \times \text{RSICARD} )</td>
</tr>
</tbody>
</table>
| Non-clustered index I matching \( \geq 1 \) preds | \( F(\text{preds}) \times (\text{NINDX(I)}+\text{NCARD})+W \times \text{RSICARD} \) 
  ...or if buffer pool large enough...
  \( F(\text{preds}) \times (\text{NINDX(I)}+\text{TCARD})+W \times \text{RSICARD} \) |
| Segment scan                     | \( \text{TCARD/P} + W \times \text{RSICARD} \) |
Roadmap - detailed

- Processing steps - overview
- Single-table query optimization
- Join query optimization
  - 2-way joins & n-way joins
  - Nested queries

Joins - Definitions

- **Outer relation** - tuple retrieved first from here
- **Inner relation** - tuples retrieved (possible based on outer tuple join value)
- **Join predicate** - relates columns of inner/outer relations

Two join methods considered

- **Nested loops** - scan inner for each outer tuple
- **Merge scans** - scan in join column order (via index or after sorting)
  - (cost formulas?)
Cost Formulae for Joins

Pi = access path

**Nested Loops:** Cost\(_{NLjoin}\) = C\(_{outer}\) (P1) + N * C\(_{inner}\) (P2)

N: # of outer tuples satisfying predicate

**Merge Join:** Cost\(_{MSjoin}\) = C\(_{outer}\) (P1) + N * C\(_{inner}\) (P2)

Since both are assumed to be sorted,
C\(_{inner}\) = #inner pages / N + W * RSICARD

Cost Formulae for Joins (cont’d)

Note: same except for C\(_{inner}\) (P2) is cheaper (potentially) in merge joins case:
Cost\(_{Sort}\) = Cost\(_{ScanPath}\) + Cost\(_{DoSortItself}\) + Cost\(_{WriteTempFile}\)

(much more accurate cost functions in Shapiro and Graefe)

N-way joins

- N! orders for N-way join (in general)
- How would you start enumerating them?
  - R1 JOIN R2 JOIN R3 JOIN R4
N-way joins (cont’d)
- N-way joins are performed as a sequence of 2-way joins
  - Can pipeline if no sort step is required
  - (Heuristic: no (R1 JOIN R2) JOIN (R3 JOIN R4))

N-way joins (cont’d)
- Cartesian products (if any) done at end
- Join orders considered only when there is an inner - outer join predicate (and outer is all relations joined so far), except if all cross-products

Example
- R1 join R2 and R2 join R3 on a different column
- Forget
  - R1 join R3 join R2
  - R3 join R1 join R2
N-way joins (cont’d)

Important observation (dynamic programming):
- After k relations have been joined, method to add in (k+1)st is independent of the order for the first k (helps organize search)

Join Optimization Algorithm

1. Find best way to access each relation for each interesting tuple order and for the unordered case
2. Best way of join any relation to these -> pairs of relations
3. Find the best way adding a third rel. to the join
4. Continue adding additional relations via step 3
5. Choose cheapest path from root to leaf

Search Tree

- Tree for possible query processing strategies:
  - Root -> leaf path represents a way of processing query
  - Label edges with costs, orderings
  - Tree considers all reasonable options
    - Access paths
    - Orderings of tuples
    - Join Orderings
  - Trees for both nested loops and merge joins
  - Always take the cheapest way for the various interesting orders and prune more expensive equivalent plans
Optimization Example

- Assume the following database schema:
  - Emp (name, dno, job, salary), indices dno (clustered), job (unclustered)
  - Dept (dno, name, loc), indices dno (clustered)
  - Job (job, title) index job (clustered)

Optimization Example (cont'd)

- Consider optimization of the following query:
  - select Emp.name, Emp.salary, Job.title, Dept.name
  - from Emp, Dept, Job
  - where title="clerk" and location="Denver"
  - and Emp.dno = Dept.dno
  - and Emp.job = Job.job

Optimization Example (cont.)

- Eligible predicates: Local predicates only
- "Interesting" orders: DNO, JOB
Access Paths for Single Relations

EMP:
- Index: EMP.DNO
  - N1: C(EMP.DNO)
- Segment Scan: EMP.JOB
  - N1: C(EMP.JOB)

DEPT:
- Segment Scan: DEPT.DNO
  - N2: C(DEPT.DNO)

JOB:
- Segment Scan: JOB.JOB
  - N3: C(JOB.JOB)

Search Tree for Single Relations

EMP:
- Index: EMP.DNO
  - N1: C(EMP.DNO)
- N1: C(EMP.JOB)

DEPT:
- Index: DEPT.DNO
  - N2: C(DEPT.DNO)
- N3: C(DEPT.JOB)

JOB:
- Segment Scan: JOB.JOB
  - N3: C(JOB.JOB)
- N3: C(JOB.SS)

Search Tree for Single Relations

EMP:
- Index: EMP.DNO
  - N1: C(EMP.DNO)
  - N1: C(EMP.JOB)

DEPT:
- Index: DEPT.DNO
  - N2: C(DEPT.DNO)
  - N3: C(DEPT.JOB)

JOB:
- Segment Scan: JOB.JOB
  - N3: C(JOB.JOB)
  - N3: C(JOB.SS)

ORDER, if any
Search Tree for Single Relations

EMP
  index EMP.DNO
  N1 C(EMP.DNO) DNO order
  COST

DEPT
  index DEPT.DNO
  N1 C(DEPT.DNO) DNO order
  N2 C(DEPT.DNO) DNO order
  N3 C(DEPT.DNO) DNO order
  JOB
    index JOB.JOB
    N3 C(JOB.JOB) JOB order
    N3 C(JOB SS) unordered

JOB
  index JOB.JOB
  N3 C(JOB SS) unordered

DEPT
  CARDINALITY

Access path

Search Tree for Single Relations

EMP
  index EMP.DNO
  N1 C(EMP.DNO) DNO order
  COST

DEPT
  index DEPT.DNO
  N1 C(DEPT.DNO) DNO order
  N2 C(DEPT.DNO) DNO order
  N3 C(DEPT.DNO) DNO order
  JOB
    index JOB.JOB
    N3 C(JOB.JOB) JOB order
    N3 C(JOB SS) unordered

JOB
  index JOB.JOB
  N3 C(JOB SS) unordered

DEPT
  CARDINALITY

Access path

Search Tree for Single Relations

EMP
  index EMP.DNO
  N1 C(EMP.DNO) DNO order
  COST

DEPT
  index DEPT.DNO
  N1 C(DEPT.DNO) DNO order
  N2 C(DEPT.DNO) DNO order
  N3 C(DEPT.DNO) DNO order
  JOB
    index JOB.JOB
    N3 C(JOB.JOB) JOB order
    N3 C(JOB SS) unordered

JOB
  index JOB.JOB
  N3 C(JOB SS) unordered

DEPT
  CARDINALITY

Access path
Next step

- Consider all 'allowed' 2-way joins - NL first

![Diagram showing 2-way joins (NLJ)]

2-way joins (NLJ)

![Diagram showing 2-way joins (MSJ)]

2-way joins (MSJ)
To add third relation

- Consider both EMP-DEPT and EMP-JOB solutions, find cheapest of each
- Then consider ways (NLJ, MSJ) to join third relation to the result

Complexity Considerations

- Exponential in N (the # of relations being joined
- Fortunately N is pretty small (<= 3) in practice
- How about # join methods considered?
- Pays off for compiled queries
- Can use heuristics for ad hoc queries
  - if the time spent optimizing exceeds the estimated execution time, quit optimizing and simply run the query

Roadmap - detailed

- Processing steps - overview
- Single-table query optimization
- Join query optimization
  - Nested queries
Nested queries

- "Uncorrelated"
  ```sql
  select name from EMP
  where dno IN ( select dno
                  from DEPT
                  where loc = "Denver");
  ```
- Q: How optimize this query?
- A: Subquery needs to be evaluated only once:
  - compute inner block first, replace cost into outer

- "Correlated"
  ```sql
  select name from EMP X
  where salary > ( select salary
                  from EMP
                  where EMP.number = X.manager)
  ```
- Must evaluate subquery for every tuple of the outer block!
- Complex (think of multiple nesting cases)
Conclusions

- Cost turns out to be good for most reasonable queries
  - Relative (not absolute) accuracy is what matters
  - proposed use of statistics (recently: better statistics)

Conclusions cont’d

- consideration of CPU utilization and I/O activity
- selectivity factors, etc
- interesting orders save unnecessary sorting
- Today: CPU costs more important
  - Need to be factored in
  - Optimization/execution times different
  - Interference

Addendum

- Uniformity and Independence assumptions
  - Neither holds!
  - Both lead to pessimistic results
    [Christodoulakis, TODS 84]

- How to avoid the uniformity assumption?
Addendum

- A: Histograms!

- For details, see [Ioannidis & Poosala, SIGMOD 95]