
Access Path Selection in Sys. R

Christos Faloutsos

www.cs.cmu.edu/~christos

Roadmap

1) Roots: System R and Ingres
2) Implementation: buffering, indexing, q-opt
   OS
   R-trees
   z-ordering
   buffering
   q-opt: joins
   system R q-opt
3) Transactions: locking, recovery
4) Distributed DBMSs
5) Parallel DBMSs: Gamma, Alphasort

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 (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 ordered needed for a
  – GroupBy,
  – OrderBy, or
  – Join

Roadmap - detailed

- Processing steps - overview
  ➪ Single-table q-opt
  – Join q-opt
  – Nested queries
Single-Relation: Cost Model

- Cost of a Query - how would you measure it?

Single-Relation: Cost Model

- Cost of a Query = \# page fetches + \( W \times \# \text{RSI Calls} \)
- \( \# \text{RSI Calls} \) = \# tuples returned by RSI
- \( W \) is a weighting factor
  - pages fetched vs. instructions executed

Single relation

- How to estimate \#I/O's, for, say
  - 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)

- \( P(T) = \frac{\text{TCARD}(T)}{\# \text{of non-empty pages in the segment}} \)
  - 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?

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>Condition</th>
<th>Formula</th>
</tr>
</thead>
</table>
| attr = value | F = 1/|ICARD(attr index)| if index exists  
              |         | F = 1/10 otherwise |
| attr1 = attr2 | F = 1/|max(|ICARD(I1),ICARD(I2))| if only index i exists, or  
            |         | F = 1/10 |
| val1 < attr < val2 | F = (value2-value1)/(high key-low key)   
                    |         | F = 1/4 otherwise |
| expr1 or expr2 | F = F(expr1)+F(expr2)−F(expr1)*F(expr2) |
| expr1 and expr2 | F = F(expr1) * F(expr2) |
| NOT expr | F = 1 − F(expr) |

Comments
• Query cardinality is the product of the relation cardinalities times the selectivities of the query’s boolean factor
  QCARD = |R_i1 |*|R_i2 |* ... * |R_in |*F_{R_1} |*F_{R_2} |* ... *F_{R_n} |
• RSICARD (# RSI calls performed) = |R_i1 |*|R_i2 |* ... * |R_in |*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*R$ICARD$
  ii) Ordering of tuples the access path will produce

Roadmap - detailed
• Processing steps - overview
• Single-table q-opt
  • Join q-opt
    – 2-way joins & n-way joins
  • Nested queries

Costs per Access Path Case

<table>
<thead>
<tr>
<th>Access Path</th>
<th>Cost Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique index matching equal predicate</td>
<td>$F + 1 + W$</td>
</tr>
<tr>
<td>Clustered index I matching $\geq 1$ preds</td>
<td>$F(preds)<em>(NINDEX(I) + TCARD) + W</em>R$ICARD$</td>
</tr>
<tr>
<td>Non-clustered index I matching $\geq 1$ preds</td>
<td>$F(preds)<em>(NINDEX(I) + NCARD) + W</em>R$ICARD$</td>
</tr>
<tr>
<td>Segment scan</td>
<td>TCARD/P + W*R$ICARD$</td>
</tr>
</tbody>
</table>

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
$P_i$ = access path

Nested Loops: $Cost_{NL, join} = C_{outer} (P1) + N* C_{inner} (P2)$
$N$ : # of outer tuples satisfying predicate

Merge Joins: $Cost_{M, join} = C_{outer} (P1) + N* C_{inner} (P2)$
Since both are assumed to be sorted,
$C_{inner} = # inner pages/N + W*R$ICARD$
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)

- do at the end the cartesian products (if any)
- 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

- Find best way to access each relation for each interesting tuple order and for the unordered case
- Best way of join any relation to these -> pairs of relations
- Find the best way adding a third rel. to the join
- Continue adding additional relations via step 3
- 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:
  ```sql
  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
  - Index EMP.JOB
  - Segment scan EMP
  - N1 C(EMP.DNO)
  - N1 C(EMP.JOB)

- **DEPT:**
  - Index DEPT.DNO
  - Segment scan DEPT
  - N2 C(DEPT.DNO)

- **JOB:**
  - Index JOB.JOB
  - Segment scan JOB
  - N3 C(JOB.JOB)

**Search Tree for Single Relations**
Search Tree for Single Relations

EMP
- Index EMP.DNO
  - N1 (EMP.DNO) DNO order

DEPT
- Index DEPT.DNO
  - N2 (DEPT.DNO) DNO order
- N3 (JOI.BJOB) Job order

JOB
- Segment scan
  - N1 (JOB.SS) unordered

ORDER, if any

COST

Cardinality

Access path

Next step

- Consider all ‘allowed’ 2-way joins - NL first

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 estimated execution time exceeds the time spent optimizing, quit optimizing and simply run the query
Roadmap - detailed

- Processing steps - overview
- **Single-table q-opt**
- Join q-opt
- Nested queries

Nested queries

- Uncorrelated: compute inner block
- ‘correlated’:
  ```sql
  select name from EMP X
  where salary > ( select salary
  from EMP
  where EMP.number = X.manager)
  ```

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 sorting unnecessarily

Addendum

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

- How to avoid the uniformity assumption?
Addendum

- **A: Histograms!**

![Salary Histogram](image)

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