Query Optimization: System-R

Selinger et al. 1979

Zachary G. Ives
University of Pennsylvania

January 27, 2003
CIS 650 – Data Sharing and the Web

Where We Are in the Semester

- First two weeks:
  - Database-style sharing, from a high level
- Next two weeks:
  - The “roots” of traditional DB query answering
  - Need to understand the “easy” problem first
- Then we start dissecting current research papers…

Administriva

- Term projects
  - Handout describing some potential projects
  - Can also suggest your own – but should meet with me this week if you want to
- Second assignment of semester:
  - Decide on a project topic
  - Do some background reading (send me mail and I'll send you refs)
  - Decide on group partner (if you want)
  - Write a 1p project proposal, due 2/5 at start of class

Looking Back at the 70s

- State of the art: COBOL, CODASYL
  - Data in hierarchical and network DBs
  - Write procedural programs to navigate them, extract desired data
- 1970-2: Codd proposed a logic-based data representation
  - Data is in sets of tuples (relations)
    - Representation-independent!!!
  - Logic-based calculus + algebra
    - Many possible orders of evaluation!!!
- CODASYL people argued:
  - Logic is too hard to write
  - RDBs couldn't be efficient

1975-9: Making the Relational Model Happen

- Friendlier languages, based on relational calculus
  - QUEL (Held et al)
  - SQL (Chamberlin et al)
- Development of new systems
  - INGRES (Berkeley), System-R (IBM San Jose)
  - Borrowed from hierarchical DBs – storage, indexing, etc.
  - Much interchange between groups (e.g., Gray, Lindsay)
  - Internally, used the relational algebra
- Key problem: performance
  - Determine best order of evaluation (and use of indices) for evaluating relational algebra expressions

Query Optimization

- Basic idea: take an algebraic expression and compile it to machine code
- System-R's key contributions:
  - Storage layer (RSS) provides a set of access paths to managed data
  - Costs of different access paths (and different algorithms) can be modeled and performance can be estimated
  - We can efficiently compare plans using a dynamic programming algorithm

RSS and Access Paths

- Tables are stored in a tuple store
  - Accessed via “segment scan” (sequential scan)
- May have B-tree indices
  - Accessed via “index scan”
  - “Sargable predicates”

- What are the cost trade-offs here?

Modeling Costs

- Every operation starts with relations on disk, outputs a relation that probably goes to disk
  - Operations have disk and CPU costs
- Begin with statistics about the data
  - Cardinality of relations; # of pages; # distinct values
- Every predicate has a selectivity factor
  - How many tuples does a predicate return, vs. the maximum possible?

What are the Factors?

- Sometimes, they're well-justified:

col = value	1 / ICARD(col index)
col1 = col2	1 / MAX(ICARD(col1 index), ICARD(col2 index))
(pred1) AND (pred2)	F(pred1) * F(pred2)

- Sometimes, they're rather arbitrary:

col = value	1 / 10 if no index
col > value	1 / 3 if non-arithmetic

Available Operators

- Selections are generally as sargable predicates
- Projection
- Sort
- Joins:
  - Nested loops – for each tuple in outer, join with all of inner
  - Merging scan (requires sorted data)
- Group by:
  - Typically done with sorted data

How Much Should They Cost?

- Depends on:
  - Indexes – selection, NL join, projection can make use of indices
  - Sorting – merging scans join, group by make use of sort order
  - Whether the output fits in memory or goes to disk

- Indexing and sorting are very different in their "downstream" effect! How?

The Heart of System-R: Optimizing Join Order

- Challenges:
  - Joins are associative and commutative (all joins are binary)
  - Two methods of doing joins
- What's a greedy recursive algorithm for doing this?
  - What are some reasons it will be inefficient?
  - What are some reasons it won't find optimal?

Dynamic Programming

- Allows us to avoid re-computation of sub-results
- Add a few heuristics:
  - Left-linear join trees
  - Postpone cartesian products
  - Selections + projections are performed as early as possible
- Example: E $\bowtie$ D $\bowtie$ J

- "Interesting orders" – why do we need them?

Contributions of System-R

- Established the basic paradigm for today's query optimizers
  - Multiple access paths to choose from
  - Offline statistics and cost model
  - Heuristics to restrict search space
  - Dynamic programming for efficiency

What Has Changed Since 1979?

- Statistics are more detailed
  - Histograms allow us to better estimate value overlap, skew, …(but only 1D histograms are commonly used)
- Cost models are more detailed
  - Generally, we can separate between random access and sequential access on a disk
- More complexity:
  - More algorithms (e.g., hash join), more operators (e.g., group by), more transformations (e.g., query decorrelation)
- Different search strategies:
  - Sometimes cartesian product is good; sometimes left-linear plans are bad

Discussion: When Does this Method Fail?