An Overview of Query Optimization in Relational Systems

Surajit Chaudhuri
Microsoft Research
surajitc@microsoft.com
http:/research.microsoft.com/~surajitc

Outline

- Preliminaries
- Query Optimization Framework
- Building Blocks
  - Equivalence Transformations
  - Statistical Model
  - *Tree-Finder: System-R, Volcano, Starburst*
- Tuning Optimizers
- Beyond the Core Optimizer
- Conclusion

System R "Tree-Finder"

- Functionality
  - Ordering among joins
  - Chooses join methods and access paths
- Naïve strategy:
  - Generate all permutations of joins
  - Generate all combinations of join methods and access paths
  - Prohibitively expensive

Exploiting Dynamic Programming

- Best plan for Join(R,S) are the same for for
  - Join (Join (R,S), T)
  - Join (Join (R,S), V)
- Optimize one sub-expression once and reuse:
  - Join (Join (Join (R,S), T), V)
  - Join (Join (Join (R,S), V), T)
- One optimal plan for each subset of relations

Example

1 2 3 4

1 2 3   1 2 4   2 3 4   134

1 2   13   1 4   23   24   3 4

Enumeration Algorithm

- Find the optimal plan for Join(T={R_1,..,R_n, R_{n+1}})
  - For each Subset S (of size n) of {R_1,..R_n, R_{n+1}} do
  - Let M = T - S
  - Find Optimal plan P(s) for Join(S)
  - Determine optimal single-join plan for Join (P(s), M)
    - Iterate over choices of join methods and access methods
    - One optimal plan for each interesting order
  - Endfor
  - Pick the plan with the least cost

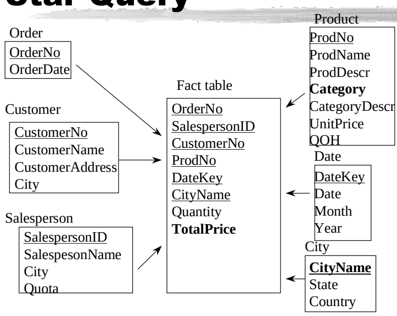
Complexity

- Enumeration cost drops from O(n!) to O(n2^n)
- May need to store O(2^n) partial plans
  - Do we?
- Significantly more efficient than the naïve scheme
- Significantly reduced number of "single-join" enumerations

Reducing Search in System-R Tree-Finder

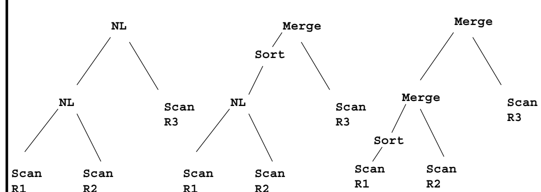
- Avoid Cartesian product
  - Defer all Cartesian products as late as possible to avoid "blow-up"
    - Don't consider (R1 X R2) Join R3 if (R1 Join R3) Join R2 is feasible
  - May result in sub-optimality
    - Large Sales Table
    - Small Store Table with selection store.loc = "Redmond"
    - Small Product Table with selection product.release = 1999
  - Recognized as "star" queries in OLAP

Star Query



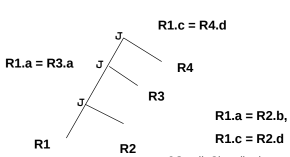
Interesting Orders

- Violation of principle of optimality



Handling Interesting Orders in Tree-Finder

- Identify all columns that may exploit sorted order (by examining join predicates)
- Collapse into equivalent groups
- One optimal partial plan for each interesting order
- Example:



Key Ideas from System R

- Cost model
- Enumeration exploits
  - Dynamic programming
  - One optimal plan for each expression
  - Violation of principle of optimality handled using interesting order
- Foundation of commercial optimizers

Limitations of System R

- Limited Transformations
  - Join ordering and choice of access methods only
  - Single block queries
- Not an adequate infrastructure for optimizing complex SQL

Outline

- Preliminaries
- Query Optimization Framework
- Building Blocks
  - Equivalence Transformations
  - Statistical Model
  - *Tree-Finder: System-R, Volcano, Starburst*
- Tuning Optimizers
- Beyond the Core Optimizer
- Conclusion

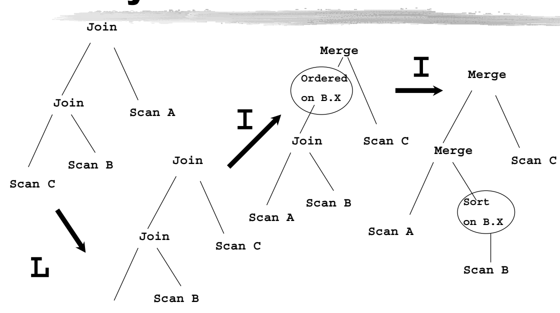
Extensible Architectures

- Extensibility for optimizer developers
  - Add arbitrary transformations
  - Add new RE operators
- Generation of the operator tree is realized as a sequence of transformations
  - What sequence of transformations will result in a *low-cost & valid* RE operator tree?
- Example: Exodus/Volcano/Cascades, Starburst
- Lets try TEA (Toy Extensible Architecture)

Plan Data Structure

- Data Structure representation of a query
  - Before Optimization: reflects query constructs (e.g., Join, Group By)
  - After Optimization: an operator tree that relational engine can execute (e.g., merge, sort)
  - During Optimization
    - Some logical operators (RA)
    - Some physical operators (RE)

Life Cycle of a Plan in TEA



Node Properties

- Logical
  - RA operator
  - Expression (sub-tree)
- Physical
  - ordering of rows
- Estimated
  - cost, total cost (sub-tree)
  - statistical descriptors

Transformation

- Condition-Action rules that
  - Preserve logical equivalence (Logical)
  - RE operator to realize a logical expression (Implementation)
  - RE operator to realize a physical property (Enforcer)
- Bind template nodes to plan nodes
  - Verify condition
  - Apply action and generate a plan

Naïve Tree-Finder

- Find (Node, Physical_Property, Cost-Limit)
  - Apply logical transformations to generate logically equivalent tree with root node'
    - Find (node', Physical_Property, Cost-Limit)
  - Apply implementation rule to a node
    - Find(child1, New1_Physical_Property, cost-limit1)
    - Find(child2, New2_Physical_Property, cost-limit2)
    - Cost = Cost1 + Cost2 + <u>Operator cost</u> (stat1,stat2)
  - Enforce a physical property

Efficiency Issues in TEA (1)

- Equivalence Classes
  - Expressions obtained by logical transformations
  - Lookup if an optimized plan exists for the class (using a hash table) or "in progress"
  - Reuse
- Use branch and bound to drive the cost limit
  - Greedy algorithms

Efficiency Issues in TEA (2)

- Rank applicable transformations
  - By promise (how?)
  - Use implementation rules only for most promising logical expression
- Top-Down Optimizer
  - Redundant optimization of sub-expressions avoided
  - Memo structure with expression (history of transformations)

Volcano v.s. Starburst

- Starburst
  - Heuristic application of logical transformation rules
  - Cost-based mapping to RE operator trees
    - Choice of access methods and join ordering for ASPJ queries
- Volcano
  - Uniformly cost-based
  - Harder to do search control

Outline

- Preliminaries
- Query Optimization Framework
- Building Blocks
  - Equivalence Transformations
  - Statistical Model
  - Tree-Finder
- *Tuning Optimizers*
- Beyond the Core Optimizer
- Conclusion

Tuning Optimizer

- Information on the plan chosen by the optimizer
  - Showplan (MS), Visual Explain (IBM) Interfaces
  - Store plan information in tables
- Optimization Level
  - How exhaustive is the search for the "optimal" plan?
    - IBM DB2: greedy v.s. DP join enumeration
- Statistics
  - Create/Update Statistics
  - Manual update to statistics

Tuning Optimizer: Hints

- Hints give partial control of execution back to the application developer
- Can specify
  - Join ordering, Join methods, Choice of Indexes
  - Example

    Select emp-id
    From Emp (index = 0)
    Where hire-date > '10/1/94'
- Liability: Hints may result in poor performance with upgrades

Outline

- Preliminaries
- Query Optimization Framework
- Building Blocks
- Tuning Optimizers
- *Beyond the Core Optimizer*
  - Parallel and Distributed Systems
  - First/Top-K Queries
  - OLAP, Materialized Views
  - Semantic Query Optimization
  - Expensive Predicates, O-R Systems, Client-Server
- Conclusion

Distributed Systems

- Optimization in Distributed Systems
  - Site Selection for operations
  - Communication cost v.s. local processing time
    - Economic Model? (Cohera)
- Evolution of Distributed Systems
  - Scalability concerns => Parallel systems
  - Distributed information => Replicated sites

Parallel Database Systems

- Objective is to minimize response time
- Forms of parallelism
  - Independent, Pipelined, Partitioned
- Issues
  - Consider Communication cost due to repartitioning
  - Scheduling of operators becomes an important aspect of optimization
  - Can/Should scheduling be separated from the rest of the query optimization?

Parallel Database Systems

- Two step approach:
  - Generate a sequential plan
  - Apply a scheduling algorithm to "parallelize" the plan
- The first phase should take into account cost of communication (e.g., repartitioning cost)
  - Influences partitioning attribute
- Scheduling algorithm assigns processors to operators
  - *Symmetric schedule:* assigns each operator equally to each processor
    - suboptimal when communication costs are considered

First/Top K Queries

- Optimize query response
  - Produce first matching record quickly
- Top k restaurants in Seattle Order by customer-satisfaction
- Optimal query plan may be different
  - Use nested loop instead of sort-merge
  - Use non-clustered index scan instead of sort
- Commercial database systems provide constructs

OLAP

- Spreadsheet paradigm drives the querying model
  - Backends always cannot digest complex SQL
  - Middleware ("ROLAP") tool optimizes SQL generation
    - Creates and maintains materialized views
    - Defines appropriate temporary relations

Materialized Views

- View Definitions
  - Aggregation as part of view definitions
  - Store Sales of Products by Quarter
- Optimization Problem
  - Materialized views enable additional logical transformations
  - Check applicability of materialized views
  - Use a *more specific* view that can answer the query
    - Sales of Products by Year is adequate to answer query on yearly revenue
- Need for a cost-based choice
  - Multiple materialized views may apply
  - Using base table may be better than using cached results

Semantic Optimization

- Transformation Rules in Classical Optimizer
  - Equivalences over SQL
- Semantic Optimization imports application knowledge
  - Constraints that hold over a database
    - No person above the age of 25
  - Optimizer can exploit these constraints

Expensive Predicates, O-R Systems

- Expensive Predicates: Cannot push down selections
  - Select * From Stocks Where stocks.*fluctuation* > .6
  - Associate a per-tuple CPU and IO cost with predicate evaluation
- O-R Systems: Relationship among udfs
  - Spatial data-blade may support related spatial indexes
  - Use rules to specify semantic relationships
  - Cost-based semantic Query Optimization
  - New issues in costing and enumeration
    - How to use costs uniformly across ADT-s
    - "Mix and match" or "ADT-specific" optimization?

Other Issues

- Compile Time v.s. Run time optimization
  - Choose plan and Exchange
- Resource governor
  - Adapting optimization to memory constraints
- Sensitivity of the cost model
- Language Extension for ease of optimization (e.g., Cube)
- Client-Server issues
  - Function shipping, Data shipping or mixed mode

Concluding Remarks

- Quality of the Optimizer depends on
  - Cost Estimator, Transformation Rules, Search Control/Enumerator
- Many external factors influence performance
  - Query Processing engine
  - Physical database design
- Oversimplification may render results useless
  - Need to pay attention to SQL semantics
- Questions?
  - Email: surajitc@microsoft.com
  - Home Page: http://research.microsoft.com/~surajitc