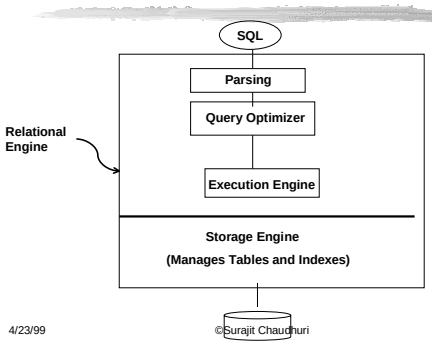
An Overview of Query Optimization in Relational Systems

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

Outline

- *Preliminaries*
  - Relational query engine
  - Operator Trees
- Query Optimization Framework
- Building Blocks
- Tuning Optimizers
- Active Areas of Research
- Conclusion

Relational DBMS Components



Scan and Selection Operators

- Scan([index], table, predicate)
  - Sequential Scan
  - Indexscan: Which index(es) to use?
  - Always push down "index-evaluable" predicates
- Filter(table, predicate)

Join Operators

- Join(Merge, R1, R2, R1.a = R2.a)
  - Requires sorted order on R1.a, R2.a
  - Output is a sorted order
  - Pipelined
- Join(Nested-Loop, R1, R2, R1.a = R2.b)
  - Sorted inputs of no consequence
  - Output has the same sort order as R1.a
  - Pipelined

"Generic" Join Operators

- Join([method], outer, inner, join-predicate)
  - Asymmetric
  - Effect of physical properties of input streams (e.g., sorted input)
  - Physical properties of output stream (e.g., sorted)
  - Pipelined v.s. Blocking (Nested Loop v.s. Sort-Merge)

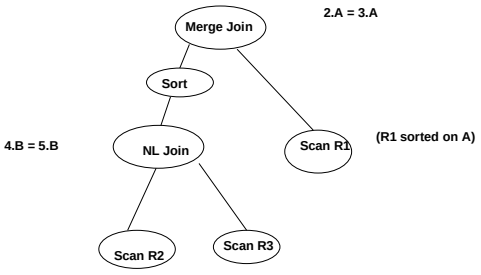
Sort & Hash

- Sort(R1, {A,B,C}, parameters)
  - Blocking (only in part)
- Build-Hash(R1, {C,D}, parameters)
  - Blocking
- Probe-Hash(R2, {C,D}, hash-table)
  - Pipelining

Generic View of RE Operators

- Input: One or more data streams
- Output: One data stream
- Implementation
  - open()
  - getnext()
  - close()
- Pipelined/Blocking

RE Operator Tree



Execution of an Operator Tree

- Demand-driven architecture is the simplest
- open() is propagated from the root
- getnext() at the root is propagated
- If getnext() at the root fails to return a new tuple, then no more answers for the query

Outline

- Preliminaries
- Query Optimization Framework
  - Why do it?
  - What information needs to be considered?
- Building Blocks
- Tuning Optimizers
- Active Areas of Research
- Conclusion

Query Optimization

- Compile a SQL query in an operator tree
- Approach 1:
  - Take a SQL expression
  - Represent as a tree
  - Turn algebraic operators into RE operators
- Why is this not adequate?

Observations

- Selections always reduce relation size
- Index scan may/may not be more efficient
- Order of evaluation of joins irrelevant for correctness
  - But, order determines efficiency
- Total sales for Products for Q298 from NW
  - Sum sales of all products, join with Products
  - Join products with Sale, then sum

Richness of Choice

- Algebraic properties allow semantically equivalent algebraic trees for a query
- Multiple implementation techniques for each algebraic operator
- Costs of the alternatives may be widely different depending on statistical properties of data

Goal of Query Optimizer

- Compile a SQL query in an operator tree
- Find the tree with least cost subject to
  - Equivalence transformations
  - Available RE operators
  - Database statistics
- Explore the space of trees efficiently
- Optimizer helps achieve
  - Data Independence

Implementing an Optimizer

- Program Optimization
- Search Algorithms/Planning
- Combinatorial Optimization
- We will revisit this issue in the next lecture..

A Framework for Query Optimization

- Equivalence Transformations
  - Algebraic properties
  - Implementation options
- Estimation Model
  - Needs to estimate cost of an operator tree (incrementally)
- Tree-Finder ("search")
  - Fast, Memory-efficient

Outline

- Preliminaries
- Query Optimization Framework
- Building Blocks
  - *Equivalence Transformations*
  - Statistical Model
  - Tree-Finder
- Tuning Optimizers
- Active Areas of Research
- Conclusion

Examples of Transformations

- Join Ordering
- Commuting join and outer-join
- Commuting group by and join
- Optimize multi-block queries
  - Collapse multi-block query to a single block query
  - Optimize across multiple query blocks

ASPJ Queries

Select A.a, Sum(B.z)
From A, B, C
Where A.x = B.x and B.y = C.y
Group By A.a
Order By A.a

Implementation Transformations

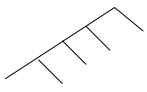
- Scan
  - B+ tree index scan with (*sargable*) Predicate: Between and its degenerate forms
  - Sequential scan
- Filter
  - Any Boolean expression
- Join
  - Sort-Merge, Nested-loop, Indexed Nested-loop

Join Ordering

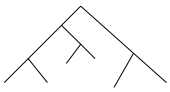
- Select and Join commute
  - Filter(Join(A,B), a) = Join(Filter(A,a), B)
- Joins are associative and commutative :
  - Join(Join(A,B), C) = Join (Join(B,A), C)
  - Join(Join(A,C), B) = Join(Join(A,B), C)
  - Many equivalent expressions (How many?)
- How about n-ary joins?

Transformation => Space

- Shape of join trees
  - Restricted use of AC properties
- Linear Join Trees



- Bushy Join Trees



Join and OuterJoin

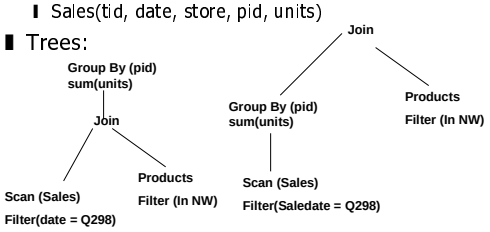
- Do outer-join operators commute?
- Example:
  - (R LOJ S): Preserves R
  - Join (R, S LOJ T) = ?
- Goal of Transformations: Isolate blocks of natural join

Group By and Join

- Select .. From .. Where .. Group By ..
  - Traditionally, execution of group-by follows execution of joins
- Total sales for Products for Q298 from NW
  - Sum sales of all products, join with Products
  - Join products with Sale, then sum

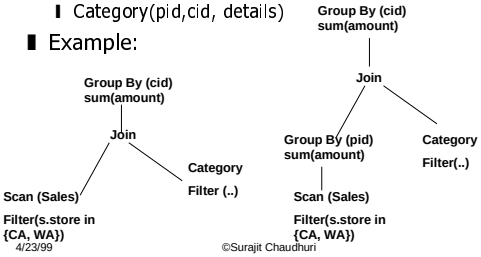
Operator Trees: Group By and Join

- Schema:
  - Product(pid, unitprice, ..)
  - Sales(tid, date, store, pid, units)
- Trees:



Introducing Operator: Group By and Join

- Schema:
  - Sales(tid, date, store, pid,amount)
  - Category(pid,cid, details)
- Example:



Prerequisites: Group By and Join

- Schema constraints, arbitrary aggregation functions
- No schema constraints, but properties of aggregate functions
  - Agg(S1 U S2) = f(Agg(S1), Agg(S2))
  - May sometime require use of derived columns

Group By and Join: Pros and Cons

- (+) "Pushing down" group by past a join:
  - Group By "collapses" an equivalence class
  - Therefore, may reduce cost of subsequent joins
  - Can be pipelined with index scans
- (-) Application needs to be cost based since
  - The cost of group by may be increased
  - Access methods on base tables may no longer be useful for the join
- Encapsulating a "transformation" is not easy

Multi-Block Queries

- Multi-block structure arises due to
  - views with aggregates
  - table expressions
  - nested sub-queries
- Techniques for Optimization
  - Merge into a single block
  - Share information across blocks