Query Optimization for XML

By Jason Mchugh, Jennifer Widom

Dazhi Wang
CPS296.1
Topics in Database Systems

Outline

- Basics for Lore system
- Motivation
- Query Execution Engine
  - Logical Query Plans
  - Statistics and Cost Models
  - Physical Plan Enumeration
- Performance Results

Introduction

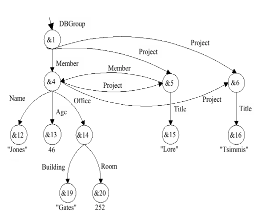
- Lore System
  - DBMS for semistructured data
  - Data model: OEM
- What complicate query optimization for XML
  - Variety of indexes and traversal techniques increases the search space
  - How to choose appropriate set of statistics for graph-based data and how to compute and store them efficiently

OEM model for XML

- OEM model: a labeled directed graph
  - Data in OEM is schema-less and self-describing
  - Atomic objects: vertices with data, no outgoing edge
  - Complex objects: vertices with outgoing edge
  - Labeled edges: express property relationships
  - Names: special labels recognizing single objects
- Correspondence between OEM and XML
  - OEM's objects: elements in XML
  - Subobject relationship in OEM: element nesting in XML
  - Difference: subelements in XML are inherently ordered; XML elements may include a list of attribute-calue pairs

OEM model for XML (cont.)

```
<Member Project = “&5 &6”>
  <Name>Jones</Name>
  <Office>
    <Building>
        Gates
    </Building>
    <Room>252</Room>
  </Office>
  <Age> 46 </Age>
</Member>
```



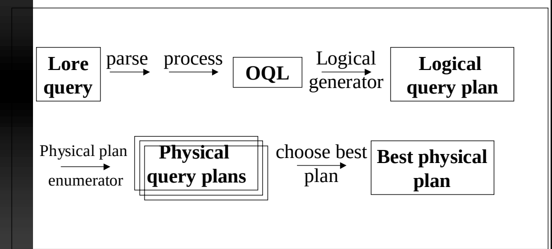
OEM model for XML (cont.)

- Simple Path Expression
  - Represents a single-step navigation in the database. Has the form *x.l y*
  - e.g. *DBGroup.Project x*
- Path Expression
  - Ordered list of simple path expressions
  - e.g. *DBGroup.Member x, x.Age y*

Lore's Query Language: *Lorel*

- Lorel is an extension of OQL, which is an SQL-like declarative language.
- Example:
```
Select x
From DBGroup.Member x
Where exists y in x.Age: y<30
```
- Operands:
  - constants, named objects, variables
- Operations:
  - Selection, set, quantification, aggregation
- Path expression can appear in *select, from* or *where* clause

Lore Query Processing



Lore Indexes

- *Vindex*
  - Find atomic objects given label and predicates
- *Lindex*
  - Find parents given object and label
- *Bindex*
  - Find parent-child pairs given an edge label
- *Pindex*
  - Find the object given a path

Path to a node is as important as the node's value

Motivation

- The optimal query plan depends on
  - *Values* in the database
  - *Shape* of the graph model

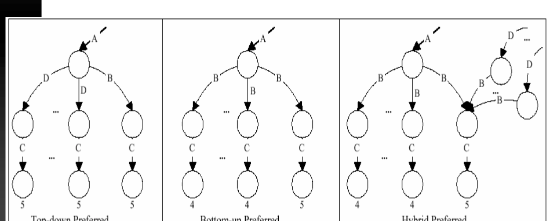
⇓

Optimization is both important and difficult

Motivation (cont.)

e.g.: Select x From A.B x where exists y in x.C: y=5

Strategy: *top-down, bottom-up, hybrid*



Query Execution Engine

- *Variable binding*: assign an object to a variable in the query
- *Evaluation*: a list of all variables in the plan along with the object bound to each variable
- The goal: iteratively generate complete evaluations to produce query results

Logical Query Plans

- To execute:

Select x From DBGroup.Member x Where exists y in x.Age: y<30

  - Top-down approach: handle *From* clause first
  - Bottom-up approach: handle *where* clause first
- For the *where* clause:
  - Break it into: (a) find all *Age* subobjects of *x*
    (b) test their values
  - Top-down approach: first (a) then (b)
  - Bottom-up approach: first find (b) using **Vindex**, then (a) using **Lindex**

Logical Query Plans (cont.)

- Solution:
  - break the query into independent components, where execution order not fixed
  - *Rotation points:* where the components meet
  - Build a tree of logical operators based on independent components
    - *Discover(x,"B", y):* bind *x* and *y*
    - *Chain(r,l):*link left & right subplan
    - *Glue(r,l):* represent a rotation point
  - Each logical operator constructs its optimal physical plan

Logical Query Plans (cont.)

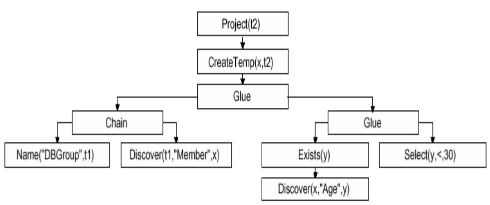


Figure 4: A complete logical query plan

A complete logical query plan for the previous example

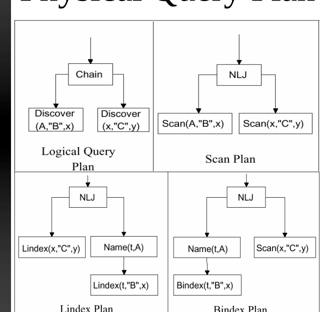
Physical Query Plan

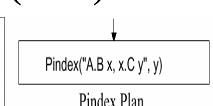
- Physical query plan operators are *iterators*
  - Request a tuple at a time from its children
  - Performs some operation
  - Returns the result to the parent
  - The "tuples" are *evaluations*
- Six basic traversal operators:
  - *Scan(**x**, l, y), Lindex(x, l, **y**), Pindex(Pathexpression, x), Bindex(l, x, y), Name(x, n), Vindex(Op, Value, l, x)*
- Other operators:
  - projection, selection, aggregation, set, etc.

Traversal Operators

- **Scan(x,l,y):** place into y all subobjects of x connected via l.
- **Lindex(x,l,y):** place into x all parents of y connected via l.
- **Pindex(pathexp, x):**
- **Bindex(l,x,y):** all x-y pairs connected via l
- **Name(x,n):** verify if x is the named obj n.
- **Vindex(Op,value,l,x):** place into x all atomic objects satisfying "Op value" and having incoming label l.

Physical Query Plan (cont.)





Subplan for:

A.B x, x.C y

Statistics and Cost Model

- **To estimate the execution cost for a given physical query plan**

Statistics

- Need to consult:
  - size & shape of the database
  - range of values
- Tradeoff between time & space and accuracy

Statistics (cont.)

- For every label subpath p of length ≤ k
  - ∀atomic type, #atomic objs (of type) reachable
  - ∀atomic type, min&max values reachable
  - Total # of instances of path *p* , $|p|$
  - Total # of distinct objs reachble, $|p|_d$
  - Total # of l-labeled sub-objects reachable, $|p|_l$
  - Total # of incoming l-labeled edges, $|p|^l$

Statistics (cont.)

- Example: evaluate `A.B x, x.C y`
  - Top-down: need to estimate # C subobjects
    - *fan-out*: $|p| * (|p|_l / |p|_d)$
  - Bottom-up: need to estimate # parents via a B edge for all C's
    - *fan-in*: $|p| * (|p|^l / |p|_d)$
- Accurate for *(k+1)* path, estimate *(k+2)* path using *(k+1)* path

Cost Model

- Execution cost:
  - I/O cost= predicted # of objects fetched
    - Rough estimation, due to lack of clustering knowledge in Lore
  - CPU cost – secondary to I/O
  - Expected # of evaluations
- Node-cost = children_cost + self_cost
- The cost model is Simplistic

Plan Enumeration

- Large search space of physical query plan
  - Path of length n ⇔ n-way join
  - Interrelated path expressions
  - Set operation, Sub-query, aggregation, etc.
- Solution: Greedy approach
  - A logical plan node makes a locally optimal decision based on bound variables passed from its parents
  - May make optimal physical subplan many times
  - Problem: still explore exponentially large # of plan

Aggressive Pruning

- Join 2 simple paths iff sharing common variable
- *Pindex* operator only starts from a named object, & no variable except the last is used elsewhere
- *Select* clause always executes last
- Separate *From* and *Where* clauses
- No reorder on multiple independent paths

n: number of simple path expressions	n=3	n=5	n=7
All possible plans/Lore's search space	1458 / 48	2,361,960 / 228	8,035,387,920 / 948

Table 1: Analysis of Search Space Size

Producing physical plans

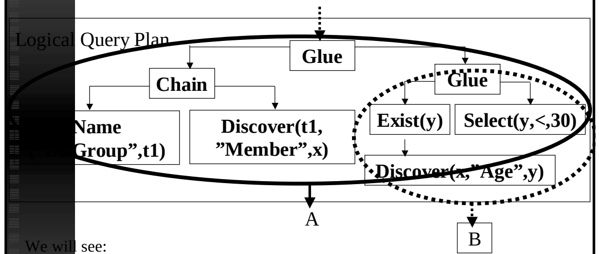
- Each logical plan node creates optimal local physical plan given a set of bound variables
- Need to track for a variable:
  - bound or not
  - Which operator bound it
  - All other operators that use it
  - Whether it is stored in a temporary result

Example: Discover(x.Age y)

x is bound: *scan* operator may be optimal

y is bound: *Lindex* operator may be optimal

Producing physical plans (Example)



We will see:

A – Left then Right

A – Right then Left → B - Left then Right

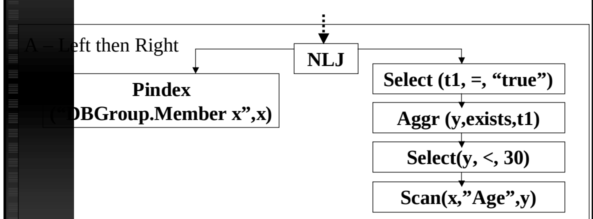
A – Right then Left → B – Right then Left

Select x

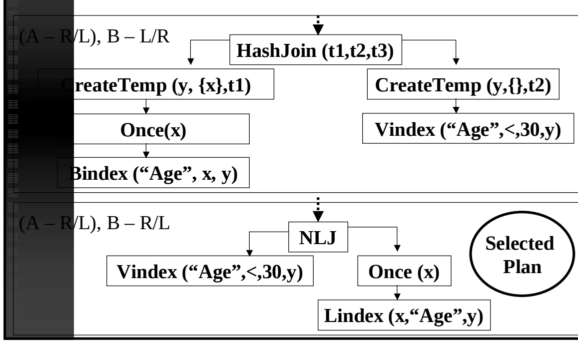
From DBGroup.Member x

Where exists y in x.Age: y<30

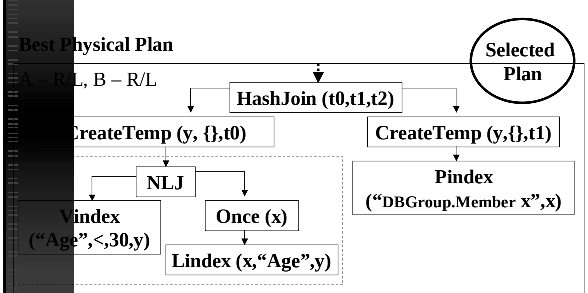
Producing physical plans (Example)



Producing physical plans (Example)



Producing physical plans (Example)



Performance Results

- Experiment over a 5 MB database on movies
- Indexes size: 8.1 MB
- 62,256 nodes
- 130,402 edges

Avg. optimization time per query ~ ½ **sec**.

Performance Results (cont.)

Plan #	1*	2	3	4	5	6	7	8	9	10	11
Exec. Time (sec.)	0.36	1.78	2.02	14.44	61.82	67.24	74.09	94.15	250.61	397.18	423.34
Estimated I/O	1975	3944	3944	9853	31918	31918	11823	37827	17742	17733	23855

Table 2: Query execution times

"Select DB.Movie.Title"

Rank	Time	Description
1*	0.3307	Bottom-up
2	0.3768	*Bindex*, *Select* then *Lindex*
7	3.3384	Top-down
24	458.58	Full *Bindex*

movies with Genre subobject having value "Comedy"

Performance Results (cont.)

Rank	Time	Description
1	0.33	Bottom-up
2*	3.68	Top-down
3	6.95	Hybrid with *Pindex*
4	7.01	Hybrid with pointer-chasing
5	23.13	*Bindex* and *Vindex* then *Lindex*

a query with two existentially quantified variables

Summary

- Logical Plan
- New Indexes
- Physical Plans
- New Statistics

Aggressive Pruning

Best physical plans