

## Relational Query Optimization

Based on:

Selinger et al. *Access Path Selection in a Relational Database Management System*, 1979

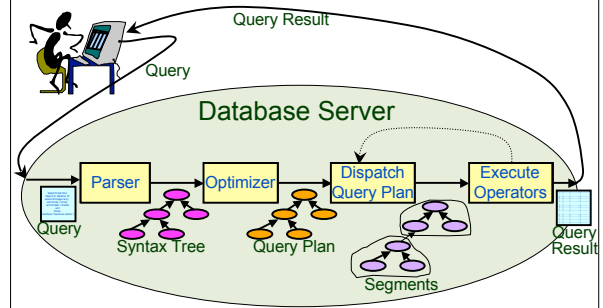
Chaudhuri. *An Overview of Query Optimization in Relational Systems*, 1998

Presentation by: Kristen LeFevre

10/10/09

1

## Relational Query Processing



10/10/09

2

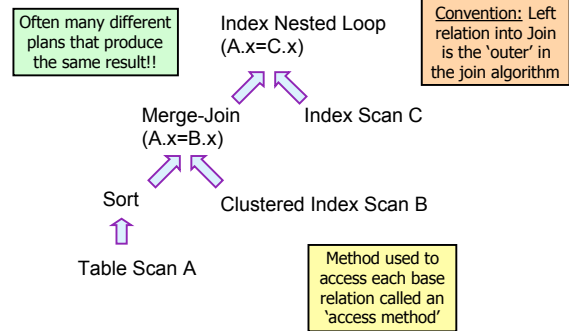
## Query Execution Plan

- Comprised of (extended) relational algebra operators
- Operator Interface: open(), getNext(), close()
- Intermediate Results (multiple ops):
  - Pipelined:** Tuples resulting from one operator fed directly into the next
  - Materialized:** Create a temporary table to store intermediate results

10/10/09

3

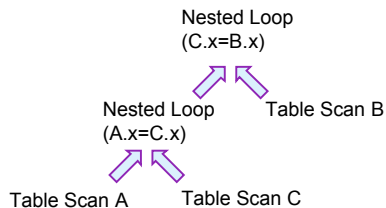
## Example



10/10/09

4

## Example -- Alternative



10/10/09

5

## Query Optimizer

- Given parsed representation of SQL query, produce an *efficient* execution plan
- Key ideas:
  - Enumerate a set of possible plans (Which ones?)
  - Estimate the cost of each plan (How?)
  - Choose the cheapest plan

10/10/09

6

## System R Optimizer

- IBM Research System (1970s)
- Still most widely-used optimization approach; works well for < 10 joins
- Access Methods (choose a method for accessing each table)
  - Index scan (clustered / unclustered)
  - Table scan
  - Selections get "pushed down" as far as possible
- Join Methods (System R considers 2)
  - Nested Loops
  - Sort-Merge

Modern systems consider additional join methods (e.g., hash join)

10/10/09 7

## System R Access Methods

Sargable predicate:  
attr <op> constant  
E.g., Age > 25

Query Operator

RDS ("Research Data System")

Sargable predicates down, tuples up

RSS ("Research Storage System")

Def: A predicate "matches" an index when it is sargable and the terms in the predicate match an initial subset of the index

10/10/09 8

## Index Matching Example

- When can we use an index to evaluate a sargable predicate?
- B+ Tree Index on (Name, Location)
- Predicates:
  - Name = 'Bob' *Matches*
  - Name = 'Bob' AND Location = 'New York' *Matches*
  - Location = 'New York' *Does not match*

10/10/09 9

## System R Statistics

- (Stored in the system catalogs)
- TCARD(T) : # pages in table T
- NCARD(T) : # tuples in table T
- P(T) : holdover from when a page could hold tuples from different relations
- ICARD(I) : # distinct keys in index
- NINDX(I) : # pages in index

10/10/09 10

## Step 1: Selectivity Estimation

- Using statistics, assign selectivity factor F for each boolean predicate
  - Very, very rough estimate!*
- attr = val
  - $F = 1/ICARD(I)$  if suitable index exists
  - 1/10 otherwise
- val1 < attr < val2
  - $F = (val2 - val1) / (highkey - lowkey)$  if index exists
  - 1/4 otherwise
- exp1 AND exp2
  - $F(exp1) * F(exp2)$
- exp1 OR exp2
  - $F(exp1) + F(exp2) - F(exp1) * F(exp2)$

10/10/09 11

## Alternatives

Came about after System R

- Histograms over a single column
  - Resolves non-uniformity problem
- Multidimensional histograms
  - Resolves correlated attributes problem
- Sampling

10/10/09 12

## Step 2: Single-Table Cost Estimation

- For each relation, calculate the cost of accessing the relation using each suitable access method (index and table scan)
- Cost = #I/Os + W \* RSI Calls  
↖ # tuples RSS returns

Example:  
 SELECT \*  
 FROM Employees  
 WHERE Name = 'Bob'  
 AND Salary = 50000

Assume:  
 Clustered index on Name  
 Unclustered index on Salary

Three Alternatives:  
 (estimate cost of each)

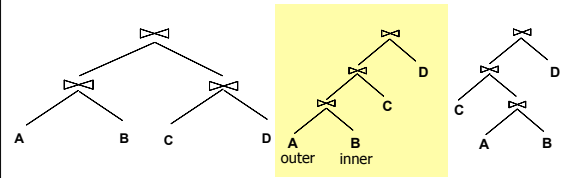
- Use Name index
- Use Salary index
- Table scan

10/10/09

13

## Step 3: Join Method and Ordering

- System R: Only consider *left-deep* join trees
  - Used to restrict the search space
  - Only left-deep plans can be *fully pipelined*.
    - Intermediate results not written to temporary files.
    - Not all left-deep trees are fully pipelined (e.g., SM join).



Linear Tree: at least 1 child in every join node is a base relation

10/10/09

14

## Enumeration of Left-Deep Plans

- Decide:
  - Join order
  - Join method for each join
- Enumerated using N passes (if N relations joined):
  - Pass 1: Find best 1-relation plan for each relation.
  - Pass 2: Find best way to join result of each 1-relation plan (as outer) to another relation. (*All 2-relation plans.*)
  - Pass N: Find best way to join result of a (N-1)-relation plan (as outer) to the N'th relation. (*All N-relation plans.*)
- For each subset of relations, retain only:
  - Cheapest plan overall, plus
  - Cheapest plan for each *interesting order* of the tuples.

10/10/09

15

## Example $\Pi_{ename} \sigma_{dname = 'Sales'} (EMP \bowtie DEPT)$

EMP (eid, ename, addr, sal, did) DEPT (did, dname, floor, mgr)

Pass 1: EMP: E1: S(EMP), E2: I(EMP.did)  
 Cost: 1000 1000+100 **KEEP E1 and E2!**  
 DEPT: D1: S(DEPT), D2: I(DEPT.did), D3: I(DEPT.dname)  
 Cost: 50 50+5 3+5 **KEEP D2 and D3**

Pass 2: Consider EMP  $\bowtie$  DEPT and DEPT  $\bowtie$  EMP  
 EMP  $\bowtie$  DEPT, Alternatives:  
 1. E1  $\bowtie$  D2: Algorithms ...  
 2. E1  $\bowtie$  D3: Algorithms ...  
 3. E2  $\bowtie$  D2: Algorithms SM, NL  
 4. E2  $\bowtie$  D3: Algorithms

Similarly consider DEPT  $\bowtie$  EMP  
 Pick cheapest 2-relation plan. Done (with join optimization)

**Next Consider GROUP BY (if present) ...**

10/10/09

16

## Additional Notes

- Only "join" relations if there is a connecting join condition i.e., *avoid Cartesian products if possible.*
- This approach is *still exponential* in the # of tables.
- ORDER BY, GROUP BY operators handled as a final step

10/10/09

17

## Nested Queries

Standard (Uncorrelated) Nested Queries:

- Evaluate nested query block first
- Use result when evaluating outer query block

```
SELECT ename
FROM EMP
WHERE did IN
  (SELECT did
   FROM DEPT
   WHERE floor = 12)
```

Correlated Nested Queries:

- In worst case, subquery must be evaluated once per tuple in outer block

```
SELECT ename
FROM EMP X
WHERE sal >
  (SELECT sal
   FROM EMP
   WHERE EMP.eid =
   X.manager)
```

10/10/09

18



## Summary

- Query optimization critical to the DBMS performance
  - Allows us to process queries expressed "declaratively"
- Main parts to optimizing a query:
  - Enumerate alternative plans
  - Estimate cost of each plan
- Single-relation queries: Pick cheapest access plan + interesting order
- Multiple-relation queries:
  - All single-relation plans are first enumerated.
  - Selections/projections considered as early as possible.
  - For each 1-relation plan, consider all ways of joining another relation (as inner)
  - Keep adding 1-relation plan until done
  - At each level, retain cheapest plan, and best plan for each interesting order