

Database System Internals

Query Optimization (part 3)

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

February 5, 2020 CSE 444 - Winter 2020 1

1

Selinger Optimizer History

- **1960's: first database systems**
 - Use tree and graph data models
- **1970: Ted Codd proposes relational model**
 - E.F. Codd. A relational model of data for large shared data banks. Communications of the ACM, 1970
- **1974: System R from IBM Research**
 - One of first systems to implement relational model
- **1979: Seminal query optimizer paper by P. Selinger et. al.**
 - Invented cost-based query optimization
 - Dynamic programming algorithm for join order computation

February 5, 2020 CSE 444 - Winter 2020 2

2

References

- P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, and T. Price. Access Path Selection in a Relational Database Management System. Proceedings of ACM SIGMOD, **1979**. Pages 22-34.

February 5, 2020 CSE 444 - Winter 2020 3

3

Selinger Algorithm

Selinger enumeration algorithm considers

- Different logical and physical plans *at the same time*
- Cost of a plan is IO + CPU
- Concept of *interesting order* during plan enumeration
 - A *sorted order* as that requested by ORDER BY or GROUP BY
 - Or order on attributes that appear in equi-join predicates
 - Because they may enable cheaper sort-merge joins later

February 5, 2020 CSE 444 - Winter 2020 4

4

More about the Selinger Algorithm

- **Step 1: Enumerate all access paths for a single relation**
 - File scan or index scan
 - Keep the cheapest for each *interesting order*
- **Step 2: Consider all ways to join two relations**
 - Use result from step 1 as the outer relation
 - Consider every other possible relation as inner relation
 - Estimate cost when using sort-merge or nested-loop join
 - Keep the cheapest for each *interesting order*
- **Steps 3 and later: Repeat for three relations, etc.**

February 5, 2020 CSE 444 - Winter 2020 5

5

Example From Selinger Paper

EMP	NAME	DNO	JOB	SAL
	SMITH	50	12	8500
	JONES	50	5	15000
	DOE	51	5	9500

DEPT	DNO	DNAME	LOC
	50	MFG	DENVER
	51	BILLING	BOULDER
	52	SHIPPING	DENVER

JOB	TITLE
5	CLERK
6	TYPIST
8	SALES
12	MECHANIC

```

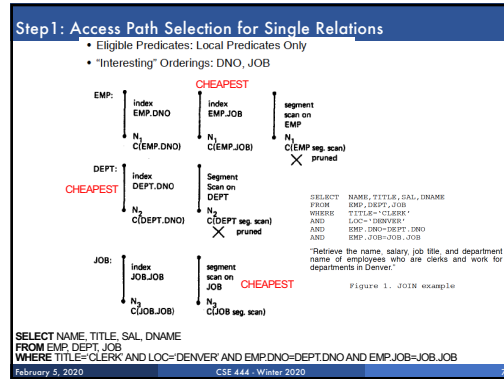
SELECT NAME, TITLE, SAL, DNAME
FROM EMP, DEPT, JOB
WHERE TITLE='CLERK'
      AND LOC='DENVER'
      AND EMP.DNO=DEPT.DNO
      AND EMP.JOB=JOB.JOB
  
```

"Retrieve the name, salary, job title, and department name of employees who are clerks and work for departments in Denver."

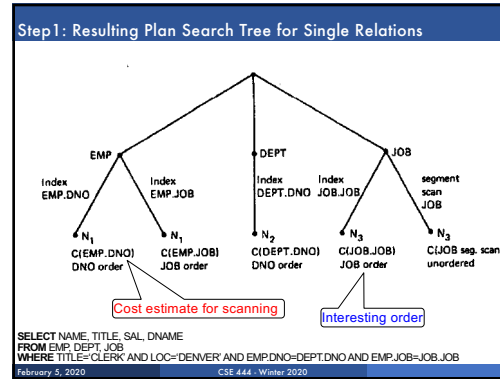
Figure 1. JOIN example

February 5, 2020 CSE 444 - Winter 2020 6

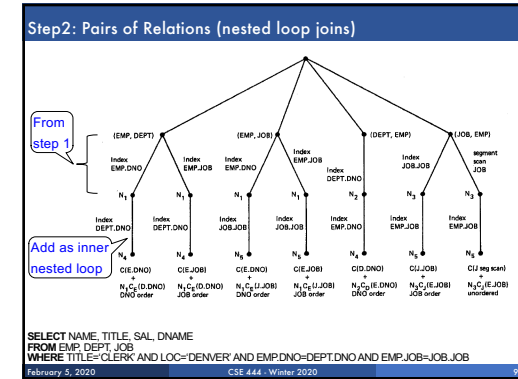
6



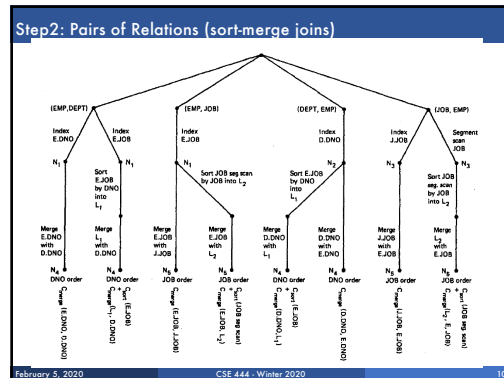
7



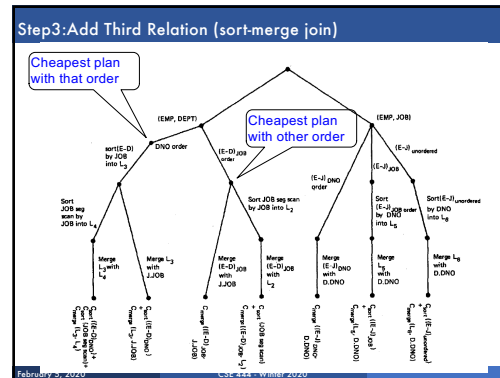
8



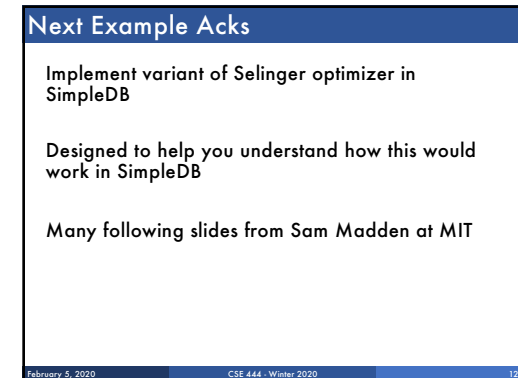
9



10



11



12

Selinger Optimizer

Problem:

- How to order a series of joins over N tables A,B,C,...
E.g. A.a = B.b AND A.c = D.d AND B.e = C.f
- NI ways to order joins; e.g. AB_{CD}, AC_{BD}, ...
- $C_{N-1} = \frac{1}{N} \binom{2(N-1)}{N-1}$ plans/ordering; e.g. (((AB)C)D), ((AB)(CD)))
- Multiple implementations (hash, nested loops)
- Naïve approach does not scale
E.g. N = 20, #join orders 20! = 2.4 x 10¹⁸; many more plans

February 5, 2020 CSE 444 - Winter 2020 13

13

Selinger Optimizer

- Only **left-deep plan**: (((AB)C)D) - eliminate C_{N-1}.
- Push down selections
- Don't consider cartesian products
- Dynamic programming algorithm

February 5, 2020 CSE 444 - Winter 2020 14

14

Dynamic Programming

OrderJoins(...):

R = set of relations to join

For d = 1 to N: /* where N = |R| */

For S in {all size-d subsets of R}:

optJoin(S) = (S - a) join a,

where a is the single relation that minimizes:

cost(**optJoin(S - a)**) +

min.cost to join (S - a) with a +

min.access cost for a

SimpleDB Lab5:
you implement **orderJoins**Use: **enumerateSubsets**Use:
computeCostAndCardOfSubplanNote: **optJoin(S-a)** is cached from previous iterations

February 5, 2020 CSE 444 - Winter 2020 15

15

Example

- orderJoins(A, B, C, D)**
- Assume all joins are Nested Loop

Subplan S	optJoin(S)	Cost(optJoin(S))
A		

February 5, 2020 CSE 444 - Winter 2020 16

16

Example

- orderJoins(A, B, C, D)**
- Assume all joins are NL

- d = 1
 - A = best way to access A (sequential scan, predicate-pushdown on index, etc)
 - B = best way to access B
 - C = best way to access C
 - D = best way to access D

- Total number of steps:
choose(N, 1)

Subplan S	optJoin(S)	Cost(optJoin(S))
A	Index scan	100
B	Seq. scan	50
C	Seq. scan	120
D	B+tree scan	400

February 5, 2020 CSE 444 - Winter 2020 17

17

Example

- orderJoins(A, B, C, D)**

- d = 2
 - {A,B} = AB or BA
use previously computed best way to access A and B

Subplan S	optJoin(S)	Cost(optJoin(S))
A	Index scan	100
B	Seq. scan	50
...		

February 5, 2020 CSE 444 - Winter 2020 18

18

Example

▪ $\text{orderJoins}(A, B, C, D)$ ▪ $d = 2$

- $\{A, B\} = AB$ or BA
use previously computed best way to access A and B

Subplan S	optJoin(S)	Cost(optJoin(S))
A	Index scan	100
B	Seq. scan	50
...		
$\{A, B\}$	BA	156

February 5, 2020

CSE 444 - Winter 2020

19

Example

▪ $\text{orderJoins}(A, B, C, D)$ ▪ $d = 2$

- $\{A, B\} = AB$ or BA
use previously computed best way to access A and B
- $\{B, C\} = BC$ or CB

Subplan S	optJoin(S)	Cost(optJoin(S))
A	Index scan	100
B	Seq. scan	50
...		
$\{A, B\}$	BA	156
$\{B, C\}$	BC	98

February 5, 2020

CSE 444 - Winter 2020

20

Example

▪ $\text{orderJoins}(A, B, C, D)$ ▪ $d = 2$

- $\{A, B\} = AB$ or BA
use previously computed best way to access A and B
- $\{B, C\} = BC$ or CB

Subplan S	optJoin(S)	Cost(optJoin(S))
A	Index scan	100
B	Seq. scan	50
...		
$\{A, B\}$	BA	156
$\{B, C\}$	BC	98

February 5, 2020

CSE 444 - Winter 2020

21

Example

▪ $\text{orderJoins}(A, B, C, D)$ ▪ $d = 2$

- $\{A, B\} = AB$ or BA
use previously computed best way to access A and B
- $\{B, C\} = BC$ or CB
- $\{C, D\} = CD$ or DC
- $\{A, C\} = AC$ or CA
- $\{B, D\} = BD$ or DB
- $\{A, D\} = AD$ or DA

Subplan S	optJoin(S)	Cost(optJoin(S))
A	Index scan	100
B	Seq. scan	50
...		
$\{A, B\}$	BA	156
$\{B, C\}$	BC	98
.....		

February 5, 2020

CSE 444 - Winter 2020

22

Example

▪ $\text{orderJoins}(A, B, C, D)$ ▪ $d = 2$

- $\{A, B\} = AB$ or BA
use previously computed best way to access A and B
- $\{B, C\} = BC$ or CB
- $\{C, D\} = CD$ or DC
- $\{A, C\} = AC$ or CA
- $\{B, D\} = BD$ or DB
- $\{A, D\} = AD$ or DA

Subplan S	optJoin(S)	Cost(optJoin(S))
A	Index scan	100
B	Seq. scan	50
...		
$\{A, B\}$	BA	156
$\{B, C\}$	BC	98
.....		

- Total number of steps: $\text{choose}(N, 2) \times 2$

February 5, 2020

CSE 444 - Winter 2020

23

Example

▪ $\text{orderJoins}(A, B, C, D)$ ▪ $d = 3$

- $\{A, B, C\} =$
Remove A: compare $A(\{B, C\})$ to $(\{B, C\})A$

Subplan S	optJoin(S)	Cost(optJoin(S))
A	Index scan	100
B	Seq. scan	50
...		
$\{A, B\}$	BA	156
$\{B, C\}$	BC	98
...		
$\{A, B, C\}$	BAC	500
.....		

February 5, 2020

CSE 444 - Winter 2020

24

Example

▪ **orderJoins(A, B, C, D)**

▪ $d = 3$

- $\{A, B, C\} =$
Remove A: compare A(**B, C**) to $\{(B, C)\}A$

Subplan S	optJoin(S)	Cost(optJoin(S))
A	Index scan	100
B	Seq. scan	50
...		
$\{A, B\}$	BA	156
$\{B, C\}$	BC	98
...		
$\{A, B, C\}$	BAC	500
.....		

optJoin(B, C) and its cost are already cached in table

February 5, 2020 CSE 444 - Winter 2020 25

25

Example

▪ **orderJoins(A, B, C, D)**

▪ $d = 3$

- $\{A, B, C\} =$
Remove A: compare A(**B, C**) to $\{(B, C)\}A$
Remove B: compare B($\{(A, C)\}$) to $\{(A, C)\}B$
Remove C: compare C($\{(A, B)\}$) to $\{(A, B)\}C$

Subplan S	optJoin(S)	Cost(optJoin(S))
A	Index scan	100
B	Seq. scan	50
...		
$\{A, B\}$	BA	156
$\{B, C\}$	BC	98
...		
$\{A, B, C\}$	BAC	500
.....		

optJoin(B, C) and its cost are already cached in table

February 5, 2020 CSE 444 - Winter 2020 26

26

Example

▪ **orderJoins(A, B, C, D)**

▪ $d = 3$

- $\{A, B, C\} =$
Remove A: compare A(**B, C**) to $\{(B, C)\}A$
Remove B: compare B($\{(A, C)\}$) to $\{(A, C)\}B$
Remove C: compare C($\{(A, B)\}$) to $\{(A, B)\}C$

Subplan S	optJoin(S)	Cost(optJoin(S))
A	Index scan	100
B	Seq. scan	50
...		
$\{A, B\}$	BA	156
$\{B, C\}$	BC	98
...		
$\{A, B, C\}$	BAC	500
.....		

optJoin(B, C) and its cost are already cached in table

February 5, 2020 CSE 444 - Winter 2020 27

27

Example

▪ **orderJoins(A, B, C, D)**

▪ $d = 3$

- $\{A, B, C\} =$
Remove A: compare A(**B, C**) to $\{(B, C)\}A$
Remove B: compare B($\{(A, C)\}$) to $\{(A, C)\}B$
Remove C: compare C($\{(A, B)\}$) to $\{(A, B)\}C$
- $\{A, B, D\} =$
Remove A: compare A($\{(B, D)\}$) to $\{(B, D)\}A$
...
- $\{A, C, D\} = \dots$
• $\{B, C, D\} = \dots$
- Total number of steps: $\text{choose}(N, 3) \times 3 \times 2$

Subplan S	optJoin(S)	Cost(optJoin(S))
A	Index scan	100
B	Seq. scan	50
...		
$\{A, B\}$	BA	156
$\{B, C\}$	BC	98
...		
$\{A, B, C\}$	BAC	500
.....		

optJoin(B, C) and its cost are already cached in table

February 5, 2020 CSE 444 - Winter 2020 28

28

Example

▪ **orderJoins(A, B, C, D)**

▪ $d = 4$

- $\{A, B, C, D\} =$
Remove A: compare A(**B, C, D**) to $\{(B, C, D)\}A$
Remove B: compare B($\{(A, C, D)\}$) to $\{(A, C, D)\}B$
Remove C: compare C($\{(A, B, D)\}$) to $\{(A, B, D)\}C$
Remove D: compare D($\{(A, B, C)\}$) to $\{(A, B, C)\}D$
- Total number of steps: $\text{choose}(N, 4) \times 4 \times 2$

Subplan S	optJoin(S)	Cost(optJoin(S))
A	Index scan	100
B	Seq. scan	50
$\{A, B\}$	BA	156
$\{B, C\}$	BC	98
$\{A, B, C\}$	BAC	500
$\{B, C, D\}$	DBC	150
.....		

optJoin(B, C, D) and its cost are already cached in table

February 5, 2020 CSE 444 - Winter 2020 29

29

Interesting Orders

- Some query plans produce data in sorted order
 - E.g. scan over a primary index, merge-join
 - Called *interesting order*
- Next operator may use this order
 - E.g. can be another merge-join
- For each subset of relations, compute multiple optimal plans, one for each interesting order
- Increases complexity by factor $k+1$, where k =number of interesting orders

February 5, 2020 CSE 444 - Winter 2020 33

33

Why Left-Deep

Asymmetric, cost depends on the order

▪ Left: Outer relation Right: Inner relation

▪ For nested-loop-join, we try to load the outer (typically smaller) relation in memory, then read the inner relation one page at a time
 $B(R) + B(R) * B(S)$ or $B(R) + B(R)/M * B(S)$

▪ For index-join, we assume right (inner) relation has index

February 5, 2020 CSE 444 - Winter 2020 34

34

Why Left-Deep

Advantages of left-deep trees?

1. Fits well with standard join algorithms (nested loop, one-pass), more efficient
2. One pass join: Uses smaller memory
 1. $((R, S), T)$: can reuse the space for R while joining (R, S) with T
 2. $(R, (S, T))$: Need to hold R, compute (S, T), then join with R, worse if more relations
3. Nested loop join, consider top-down iterator next()
 1. $((R, S), T)$: Reads the chunks of (R, S) once, reads stored base relation T multiple times
 2. $(R, (S, T))$: Reads the chunks of R once, reads computed relation (S, T) multiple times, either more time or more space

February 5, 2020 CSE 444 - Winter 2020 35

35

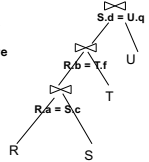
Implementation in SimpleDB (lab5)

1. `JoinOptimizer.java` (and the classes used there)

2. Returns vector of "LogicalJoinNode"

Two base tables, two join attributes, predicate e.g. $R(a, b), S(c, d), T(a, f), U(p, q)$
 $(R, S, R.a, S.c, =)$
 Recall that SimpleDB keeps all attributes of R, S after their join $R.a, R.b, S.c, S.d$

3. Output vector looks like:
 $\langle (R, S, R.a, S.c), (R, T, R.b, T.f), (S, U, S.d, U.q) \rangle$



February 5, 2020 CSE 444 - Winter 2020 36

36

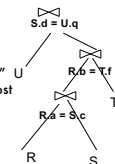
Implementation in SimpleDB (lab5)

Any advantage of returning pairs?

▪ Flexibility to consider all linear plans
 $\langle (R, S, R.a, S.c), (R, T, R.b, T.f), (U, S, U.q, S.d) \rangle$

More Details:

1. You mainly need to implement "orderJoins(...)"
2. "CostCard" data structure stores a plan, its cost and cardinality: you would need to estimate them
3. "PlanCache" stores the table in dyn. Prog:
 Maps a set of LIN to a vector of LIN (best plan for the vector), its cost, and its cardinality
LIN = LogicalJoinNode



February 5, 2020 CSE 444 - Winter 2020 37

37