

CSE 444: Database Internals

Lecture 12 Query Optimization (part 3)

CSE 444 - Spring 2019

1

Announcements

CSE 444 - Spring 2019

2

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

CSE 444 - Spring 2019

3

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.

CSE 444 - Spring 2019

4

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

CSE 444 - Spring 2019

5

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.

CSE 444 - Spring 2019

6

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	JOB	TITLE
	5	CLERK
	6	TYPIST
	8	SALES
	12	MECHANIC

```

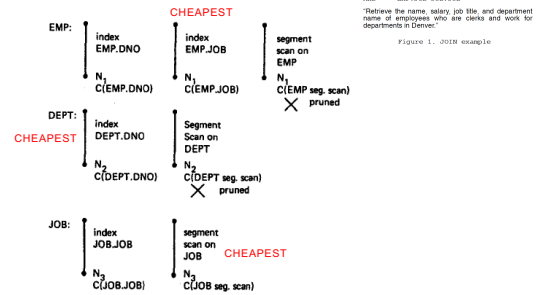
SELECT NAME, TITLE, SAL, DNAME
FROM EMP, DEPT, JOB
WHERE TITLE='CLERK'
AND LOC='DENVER'
AND EMP.DNO=DEPT.DNO
AND EMP.JOB=JOB.JOB
    
```

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

Figure 1. JOIN example

Step1: Access Path Selection for Single Relations

- Eligible Predicates: Local Predicates Only
- "Interesting" Orderings: DNO, JOB

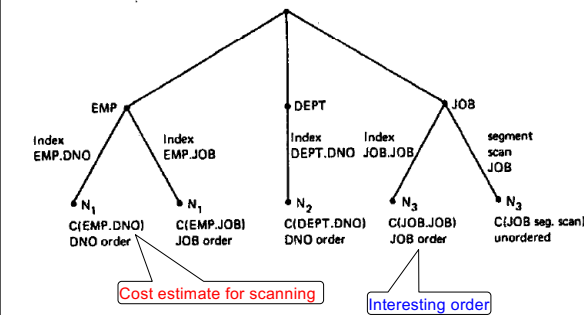


```

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
    
```

8

Step1: Access Path Selection for Single Relations Resulting Plan Search Tree for Single Relations

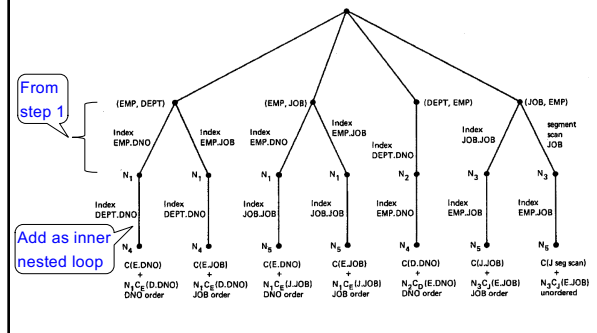


```

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
    
```

9

Step2: Pairs of Relations (nested loop joins)

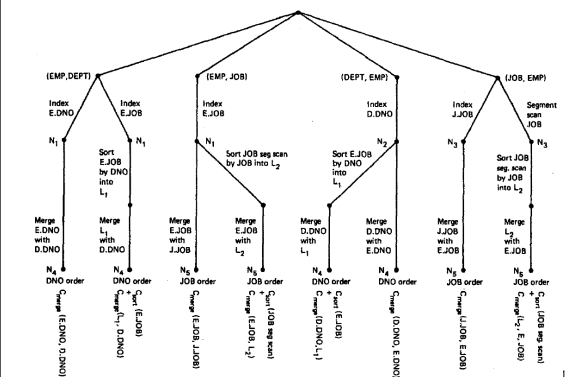


```

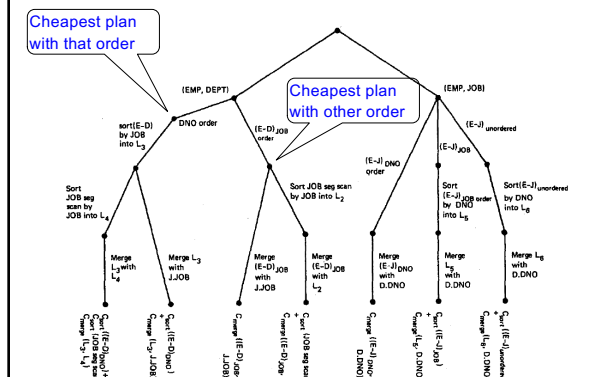
SELECT NAME, TITLE, SAL, DNAME
FROM EMP, DEPT, JOB
WHERE TITLE='CLERK' AND LOC='DENVER' AND EMP.DNO=DEPT.DNO AND EMP.JOB=JOB.JOB
    
```

10

Step2: Pairs of Relations (sort-merge joins)



Step3: Add Third Relation (sort-merge join)



Next Example Acks

Implement variant of Selinger optimizer in SimpleDB

Designed to help you with Lab 5

Many following slides from Sam Madden at MIT

CSE 444 - Spring 2019

13

Dynamic Programming

OrderJoins(...):

SimpleDB Lab5:
you implement **orderJoins**

R = set of relations to join

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

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

Use: **enumerateSubsets**

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

Use:
computeCostAndCardOfSubplan

Note: **optjoin**(S-a) is cached from previous iterations

CSE 444 - Spring 2019

16

Example

• **orderJoins(A, B, C, D)**

• Assume all joins are Nested Loop

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

CSE 444 - Spring 2019

17

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

CSE 444 - Spring 2019

18

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
...		

CSE 444 - Spring 2019

19

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
...		
{A, B}	BA	156

CSE 444 - Spring 2019

20

Example

- **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

CSE 444 - Spring 2019

21

Example

- **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

CSE 444 - Spring 2019

22

Example

- **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
.....		

CSE 444 - Spring 2019

23

Example

- **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: $\text{choose}(N, 2) \times 2$

CSE 444 - Spring 2019

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
.....		

CSE 444 - Spring 2019

25

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
.....		

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

CSE 444 - Spring 2019

26

Example

- $\text{orderJoins}(A, B, C, D)$

- $d = 3$

– $\{A, B, C\} =$
 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

CSE 444 - Spring 2019

27

Example

- $\text{orderJoins}(A, B, C, D)$

- $d = 3$

– $\{A, B, C\} =$
 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

CSE 444 - Spring 2019

28

Example

- $\text{orderJoins}(A, B, C, D)$

- $d = 3$

– $\{A, B, C\} =$
 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

– $\{A, B, D\} =$
 Remove A: compare $A(\{B, D\})$ to $\{(B, D)\}A$

– $\{A, C, D\} = \dots$
 – $\{B, C, D\} = \dots$

- Total number of steps: $\text{choose}(N, 3) \times 3 \times 2$

29

Example

- $\text{orderJoins}(A, B, C, D)$

- $d = 4$

– $\{A, B, C, D\} =$

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

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: $\text{choose}(N, 4) \times 4 \times 2$

30

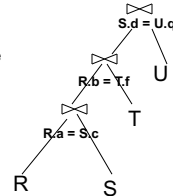
Implementation in SimpleDB (lab5)

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:

$\langle (R, S, R.a, S.c), (R, T, R.b, T.f), (S, U, S.d, U.q) \rangle$

CSE 444 - Spring 2019

37

Implementation in SimpleDB (lab5)

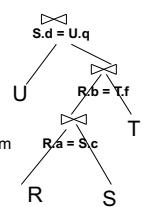
Any advantage of returning pairs?

- 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 LJO to a vector of LJO (best plan for the vector), its cost, and its cardinality
LJO = LogicalJoinNode



CSE 444 - Spring 2019

38