

## CSE 444: Database Internals

### Lecture 11

#### Query Optimization (part 2)

CSE 444 - Spring 2019

1

## Two Types of Optimizers

- **Heuristic-based optimizers:**

- Apply greedily rules that always improve plan
  - Typically: push selections down
  - Very limited: no longer used today

- **Cost-based optimizers:**

- Use a cost model to estimate the cost of each plan
  - Select the “cheapest” plan
  - We focus on cost-based optimizers

CSE 444 - Spring 2019

3

## The Three Parts of an Optimizer

- Cost estimation
  - Based on cardinality estimation
- Search space
- Search algorithm

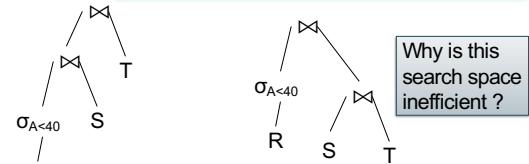
CSE 444 - Spring 2019

4

## Complete Plans

$R(A,B)$   
 $S(B,C)$   
 $T(C,D)$

`SELECT *  
 FROM R, S, T  
 WHERE R.B=S.B and S.C=T.C and R.A<40`



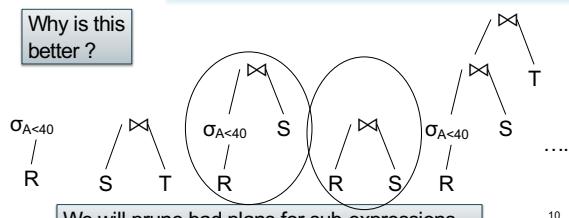
9

## Bottom-up Partial Plans

$R(A,B)$   
 $S(B,C)$   
 $T(C,D)$

`SELECT *  
 FROM R, S, T  
 WHERE R.B=S.B and S.C=T.C and R.A<40`

Why is this better?

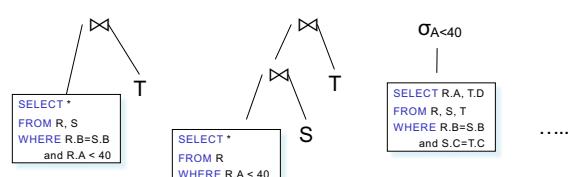


10

## Top-down Partial Plans

$R(A,B)$   
 $S(B,C)$   
 $T(C,D)$

`SELECT *  
 FROM R, S, T  
 WHERE R.B=S.B and S.C=T.C and R.A<40`



CSE 444 - Spring 2019

11

## Query Optimizer Overview

- **Input:** A logical query plan
- **Output:** A good physical query plan
- **Basic query optimization algorithm**
  - Enumerate alternative plans (logical and physical)
  - Compute estimated cost of each plan
    - Compute number of I/Os
    - Optionally take into account other resources
  - Choose plan with lowest cost
  - This is called cost-based optimization

CSE 444 - Spring 2019

12

## The Three Parts of an Optimizer

- Cost estimation
  - Based on cardinality estimation
- Search space
  - Algebraic laws, restricted types of join trees
- **Search algorithm**
  - Will discuss next

CSE 444 - Spring 2019

13

## Search Algorithm

- Dynamic programming ([in class](#))
  - Based on System R (aka Selinger) style optimizer[1979]
  - Limited to joins: *join reordering algorithm*
  - [Bottom-up](#)
- Rule-based algorithm ([will not discuss](#))
  - Database of rules (=algebraic laws)
  - Usually: dynamic programming
  - Usually: [top-down](#)

CSE 444 - Spring 2019

14

## Dynamic Programming

Originally proposed in System R [1979]

- Only handles single block queries:

```
SELECT list
FROM   R1, ..., Rn
WHERE cond1 AND cond2 AND ... AND condk
```

- Some heuristics for search space enumeration:
  - Selections down
  - Projections up
  - Avoid cartesian products

CSE 444 - Spring 2019

15

## Dynamic Programming

- For each subquery  $Q \subseteq \{R_1, \dots, R_n\}$  compute the following:
  - $T(Q)$  = the estimated size of  $Q$
  - $\text{Plan}(Q)$  = a best plan for  $Q$
  - $\text{Cost}(Q)$  = the estimated cost of that plan

CSE 444 - Spring 2019

16

## Dynamic Programming

- **Step 1:** For each  $\{R_i\}$  do:
  - $T(\{R_i\}) = T(R_i)$
  - $\text{Plan}(\{R_i\})$  = access method for  $R_i$
  - $\text{Cost}(\{R_i\})$  = cost of access method for  $R_i$

CSE 444 - Spring 2019

17

## Dynamic Programming

- **Step 2:** For each  $Q \subseteq \{R_1, \dots, R_n\}$  of size  $k$  do:

- $T(Q)$  = use estimator
- Consider all partitions  $Q = Q' \cup Q''$   
compute cost(Plan( $Q'$ )  $\bowtie$  Plan( $Q''$ ))
- Cost( $Q$ ) = the smallest such cost
- Plan( $Q$ ) = the corresponding plan

- Note

- If we restrict to left-linear trees:  $Q''$  = single relation
- May want to avoid cartesian products

CSE 444 - Spring 2019

18

## Dynamic Programming

- **Step 3:** Return Plan( $\{R_1, \dots, R_n\}$ )

CSE 444 - Spring 2019

19

## Example

- $R \bowtie S \bowtie T \bowtie U$
- Assumptions:
 

$T(R) = 2000$
$T(S) = 5000$
$T(T) = 3000$
$T(U) = 1000$
- Every join selectivity is 0.001

CSE 444 - Spring 2019

20

SELECT \*  
FROM R, S, T, U  
WHERE cond1 AND cond2 AND ...

$T(R) = 2000$   
 $T(S) = 5000$   
 $T(T) = 3000$   
 $T(U) = 1000$

Assume  
 $B(..) = T(..)/10$

Join selectivity  
is 0.001

Subquery	T	Plan	Cost
R	2000		
S	5000		
T	3000		
U	1000		
RS			
RT			
RU			
ST			
SU			
TU			
RST			
RSU			
RTU			
STU			
RSTU			

21

$T(R) = 2000$   
 $T(S) = 5000$   
 $T(T) = 3000$   
 $T(U) = 1000$

Assume  
 $B(..) = T(..)/10$

Join selectivity  
is 0.001

Subquery	T	Plan	Cost
R	2000		
S	5000		
T	3000		
U	1000		
RS	10000		
RT	6000		
RU	2000		
ST	15000		
SU	5000		
TU	3000		
RST	30000		
RSU	10000		
RTU	6000		
STU	15000		
RSTU	30000		

22

$T(R) = 2000$   
 $T(S) = 5000$   
 $T(T) = 3000$   
 $T(U) = 1000$

Assume  
 $B(..) = T(..)/10$

Join selectivity  
is 0.001

Subquery	T	Plan	Cost
R	2000	Clustered index scan R.A	200
S	5000		
T	3000		
U	1000		
RS	10000		
RT	6000		
RU	2000		
ST	15000		
SU	5000		
TU	3000		
RST	30000		
RSU	10000		
RTU	6000		
STU	15000		
RSTU	30000		

23

$T(R) = 2000$   
 $T(S) = 5000$   
 $T(T) = 3000$   
 $T(U) = 1000$

Assume  $B(\cdot) = T(\cdot)/10$

Join selectivity is 0.001

Subquery	T	Plan	Cost
R	2000	Clustered index scan R.A	200
S	5000	Table scan	500
T	3000		
U	1000		
RS	10000		
RT	6000		
RU	2000		
ST	15000		
SU	5000		
TU	3000		
RST	30000		
RSU	10000		
RTU	6000		
STU	15000		
RSTU	30000		

24

$T(R) = 2000$   
 $T(S) = 5000$   
 $T(T) = 3000$   
 $T(U) = 1000$

Assume  $B(\cdot) = T(\cdot)/10$

Join selectivity is 0.001

Subquery	T	Plan	Cost
R	2000	Clustered index scan R.A	200
S	5000	Table scan	500
T	3000	Table scan	300
U	1000	Unclustered index scan U.F	1000
RS	10000		
RT	6000		
RU	2000		
ST	15000		
SU	5000		
TU	3000		
RST	30000		
RSU	10000		
RTU	6000		
STU	15000		
RSTU	30000		

25

$T(R) = 2000$   
 $T(S) = 5000$   
 $T(T) = 3000$   
 $T(U) = 1000$

Assume  $B(\cdot) = T(\cdot)/10$

Join selectivity is 0.001

Subquery	T	Plan	Cost
R	2000	Clustered index scan R.A	200
S	5000	Table scan	500
T	3000	Table scan	300
U	1000	Unclustered index scan U.F	1000
RS	10000	R $\bowtie$ S nested loop join	...
RT	6000		
RU	2000		
ST	15000		
SU	5000		
TU	3000		
RST	30000		
RSU	10000		
RTU	6000		
STU	15000		
RSTU	30000		

26

$T(R) = 2000$   
 $T(S) = 5000$   
 $T(T) = 3000$   
 $T(U) = 1000$

Assume  $B(\cdot) = T(\cdot)/10$

Join selectivity is 0.001

Subquery	T	Plan	Cost
R	2000	Clustered index scan R.A	200
S	5000	Table scan	500
T	3000	Table scan	300
U	1000	Unclustered index scan U.F	1000
RS	10000	R $\bowtie$ S nested loop join	...
RT	6000	R $\bowtie$ T index join	...
RU	2000		
ST	15000		
SU	5000		
TU	3000		
RST	30000		
RSU	10000		
RTU	6000		
STU	15000		
RSTU	30000		

27

$T(R) = 2000$   
 $T(S) = 5000$   
 $T(T) = 3000$   
 $T(U) = 1000$

Assume  $B(\cdot) = T(\cdot)/10$

Join selectivity is 0.001

Subquery	T	Plan	Cost
R	2000	Clustered index scan R.A	200
S	5000	Table scan	500
T	3000	Table scan	300
U	1000	Unclustered index scan U.F	1000
RS	10000	R $\bowtie$ S nested loop join	...
RT	6000	R $\bowtie$ T index join	...
RU	2000	R $\bowtie$ U index join	
ST	15000	S $\bowtie$ T hash join	
SU	5000	...	
TU	3000	...	
RST	30000		
RSU	10000		
RTU	6000		
STU	15000		
RSTU	30000		

28

$T(R) = 2000$   
 $T(S) = 5000$   
 $T(T) = 3000$   
 $T(U) = 1000$

Assume  $B(\cdot) = T(\cdot)/10$

Join selectivity is 0.001

Subquery	T	Plan	Cost
R	2000	Clustered index scan R.A	200
S	5000	Table scan	500
T	3000	Table scan	300
U	1000	Unclustered index scan U.F	1000
RS	10000	R $\bowtie$ S nested loop join	...
RT	6000	R $\bowtie$ T index join	...
RU	2000	R $\bowtie$ U index join	
ST	15000	S $\bowtie$ T hash join	
SU	5000	...	
TU	3000	...	
RST	30000	(RT) $\bowtie$ S hash join	...
RSU	10000	(SU) $\bowtie$ R merge join	
RTU	6000	...	
STU	15000		
RSTU	30000		

29

$T(R) = 2000$	T	Plan	Cost
$T(S) = 5000$	R	2000	Clustered index scan R.A
$T(T) = 3000$	S	5000	Table scan
$T(U) = 1000$	T	3000	Table scan
Assume $B(..) = T(..)/10$	U	1000	Unclustered index scan U.F
Join selectivity is 0.001	RS	10000	$R \bowtie S$ nested loop join
	RT	6000	$R \bowtie T$ index join
	RU	2000	$R \bowtie U$ index join
	ST	15000	$S \bowtie T$ hash join
	SU	5000	...
	TU	3000	...
	RST	30000	$(RT) \bowtie S$ hash join
	RSU	10000	$(SU) \bowtie R$ merge join
	RTU	6000	...
	STU	15000	...
	RSTU	30000	$(RT) \bowtie (SU)$ hash join
		...	

30

## Discussion

- For the subset RS, need to consider both  $R \bowtie S$  and  $S \bowtie R$ 
  - Because the cost may be different!
- When computing the cheapest plan for  $(Q) \bowtie R$ , we may consider new access methods for R, e.g. an index look-up that makes sense only in the context of the join

CSE 444 - Spring 2019

31

## Discussion

Given a query with n relations  $R_1, \dots, R_n$

- How many entries do we have in the dynamic programming table?
- For each entry, how many alternative plans do we need to inspect?

CSE 444 - Spring 2019

32

## Discussion

Given a query with n relations  $R_1, \dots, R_n$

- How many entries do we have in the dynamic programming table?
  - A:  $2^n - 1$
- For each entry, how many alternative plans do we need to inspect?
  - A: for each entry with k tables, examine  $2^k - 2$  plans

CSE 444 - Spring 2019

33

## Reducing the Search Space

- Left-linear trees
- No cartesian products

CSE 444 - Spring 2019

34

## Join Trees

- $R_1 \bowtie R_2 \bowtie \dots \bowtie R_n$
  - Join tree:
- 
- ```

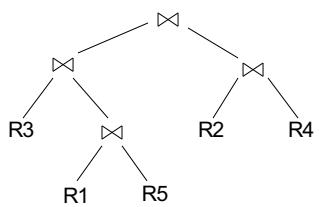
graph TD
    Root(( )) --> R1((R1))
    Root(( )) --> R2((R2))
    R1((R1)) --> R3((R3))
    R1((R1)) --> R4((R4))
    R2((R2)) --> R4((R4))
  
```
- A plan = a join tree
  - A partial plan = a subtree of a join tree

CSE 444 - Spring 2019

35

## Types of Join Trees

- Bushy:

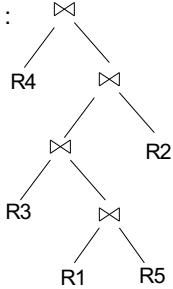


CSE 444 - Spring 2019

36

## Types of Join Trees

- Linear :

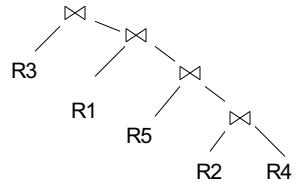


CSE 444 - Spring 2019

37

## Types of Join Trees

- Right deep:



CSE 444 - Spring 2019

38

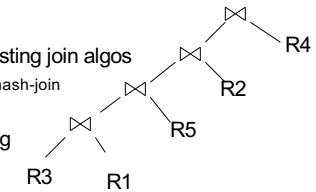
## Types of Join Trees

- Left deep:

– Work well with existing join algos

- Nested-loop and hash-join

– Facilitate pipelining



– Dynamic programming can be used with all trees

CSE 444 - Spring 2019

39