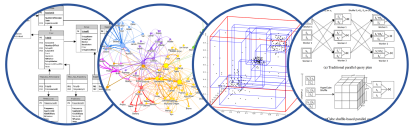


Database System Internals

Query Optimization (part 2)

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

1

Announcements

- HW 2 due tonight
- Lab 1 grades and feedback back later this week
- Quiz 1+2 on Feb. 12th
  - Not as long as a midterm
  - Examples will be posted on webpage

2

Two Types of Plan Enumeration Algorithms

- 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

3

Two Types of Optimizers

- Rule-based (heuristic) 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

4

The Three Parts of an Optimizer

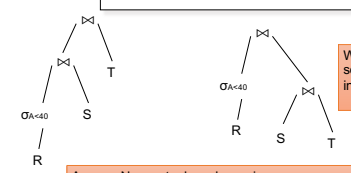
- Cost estimation
  - Based on cardinality estimation
- Search space
- Search algorithm

5

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
```



Why is this search space inefficient ?

Answer: No way to do early pruning

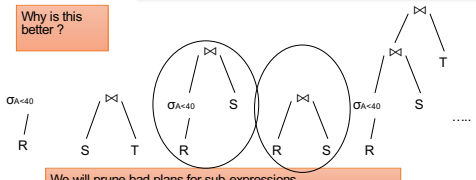
10

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 ?



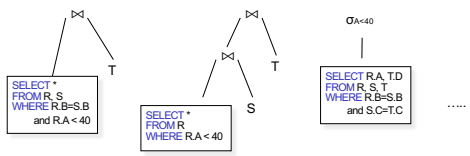
We will prune bad plans for sub-expressions

11

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
```



12

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

13

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

14

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

15

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

16

Dynamic Programming

SELECT list
FROM R1, …, Rn
WHERE $cond_1$ AND $cond_2$ AND . . . AND $cond_k$

- For each subquery Q ⊆{R1, …, Rn} compute the following:
  - T(Q) = the estimated size of Q
  - Plan(Q) = a best plan for Q
  - Cost(Q) = the estimated cost of that plan

17

Dynamic Programming

SELECT list
FROM R1, …, Rn
WHERE $cond_1$ AND $cond_2$ AND . . . AND $cond_k$

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

18

Dynamic Programming

SELECT list
FROM R1, …, Rn
WHERE $cond_1$ AND $cond_2$ AND . . . AND $cond_k$

- **Step 2**: For each Q ⊆$\{R_1, \ldots, R_n\}$ of size k do:
  - T(Q) = use estimator
  - Consider all partitions Q = Q' ∪ Q'' compute cost(Plan(Q') ⋈ 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

19

Dynamic Programming

SELECT list
FROM R1, …, Rn
WHERE $cond_1$ AND $cond_2$ AND . . . AND $cond_k$

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

20

Example

SELECT *
FROM R, S, T, U
WHERE $cond_1$ AND $cond_2$ AND . . .

- R ⋈ S ⋈ T ⋈ U
- Assumptions:

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

- Every join selectivity is 0.001

21

Example

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			

22

Example

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		

23

Example

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		

24

Example

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	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		

25

Example

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	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		

26

Example

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	Table scan	500
T	3000	Table scan	300
U	1000	**Unclustered index scan U.F**	1000
RS	10000	R ⋈ S nested loop join	…
RT	6000		
RU	2000		
ST	15000		
SU	5000		
TU	3000		
RST	30000		
RSU	10000		
RTU	6000		
STU	15000		
RSTU	30000		

27

Example

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	Table scan	500
T	3000	Table scan	300
U	1000	**Unclustered index scan U.F**	1000
RS	10000	R ⋈ S nested loop join	…
RT	6000	**R ⋈ T index** join	…
RU	2000		
ST	15000		
SU	5000		
TU	3000		
RST	30000		
RSU	10000		
RTU	6000		
STU	15000		
RSTU	30000		

28

Example

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	Table scan	500
T	3000	Table scan	300
U	1000	Unclustered index scan U.F	1000
RS	10000	R ⋈ S nested loop join	...
RT	6000	R ⋈ T index join	...
RU	2000	R ⋈ U index join	
ST	15000	S ⋈ T hash join	
SU	5000	. . .	
TU	3000	. . .	
RST	30000		
RSU	10000		
RTU	6000		
STU	15000		
RSTU	30000		

29

Example

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	Table scan	500
T	3000	Table scan	300
U	1000	Unclustered index scan U.F	1000
RS	10000	R ⋈ S nested loop join	...
RT	6000	R ⋈ T index join	...
RU	2000	R ⋈ U index join	
ST	15000	S ⋈ T hash join	
SU	5000	. . .	
TU	3000	. . .	
RST	30000	(RT) ⋈ S hash join	...
RSU	10000	(SU) ⋈ R merge join	
RTU	6000	...	
STU	15000		
RSTU	30000		

30

Example

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	Table scan	500
T	3000	Table scan	300
U	1000	Unclustered index scan U.F	1000
RS	10000	R ⋈ S nested loop join	...
RT	6000	R ⋈ T index join	...
RU	2000	R ⋈ U index join	
ST	15000	S ⋈ T hash join	
SU	5000	. . .	
TU	3000	. . .	
RST	30000	(RT) ⋈ S hash join	...
RSU	10000	(SU) ⋈ R merge join	
RTU	6000	...	
STU	15000		
RSTU	30000	(RT) ⋈ (SU) hash join	...

31

Discussion

- For the subset {RS}, need to consider both
  R ⋈ S and S ⋈ R
  - Because the cost may be different!
- When computing the cheapest plan for
  (Q) ⋈ 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

32

Discussion

SELECT list
FROM R1, ..., Rn
WHERE $cond_1$ AND $cond_2$ AND . . . AND $cond_k$

Given a query with n relations R1, ..., Rn

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

33

Discussion

SELECT list
FROM R1, ..., Rn
WHERE $cond_1$ AND $cond_2$ AND . . . AND $cond_k$

Given a query with n relations R1, ..., Rn

- 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

34

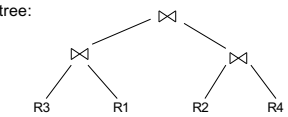
Reducing the Search Space

- Left-linear trees
- No cartesian products

35

Join Trees

- R1 ⋈ R2 ⋈ …. ⋈ Rn
- Join tree:

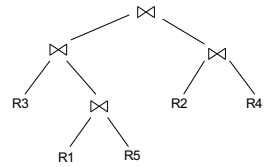


- A plan = a join tree
- A partial plan = a subtree of a join tree

36

Types of Join Trees

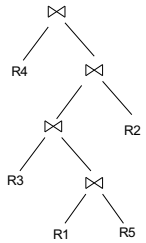
- Bushy:



37

Types of Join Trees

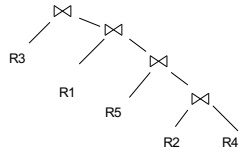
- Linear :



38

Types of Join Trees

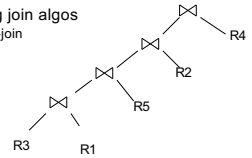
- Right deep:



39

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

40

No Cartesian Products

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

Plan: $(R(A,B) \bowtie T(C,D)) \bowtie S(B,C)$
has a cartesian product.
Most query optimizers will not consider it

42