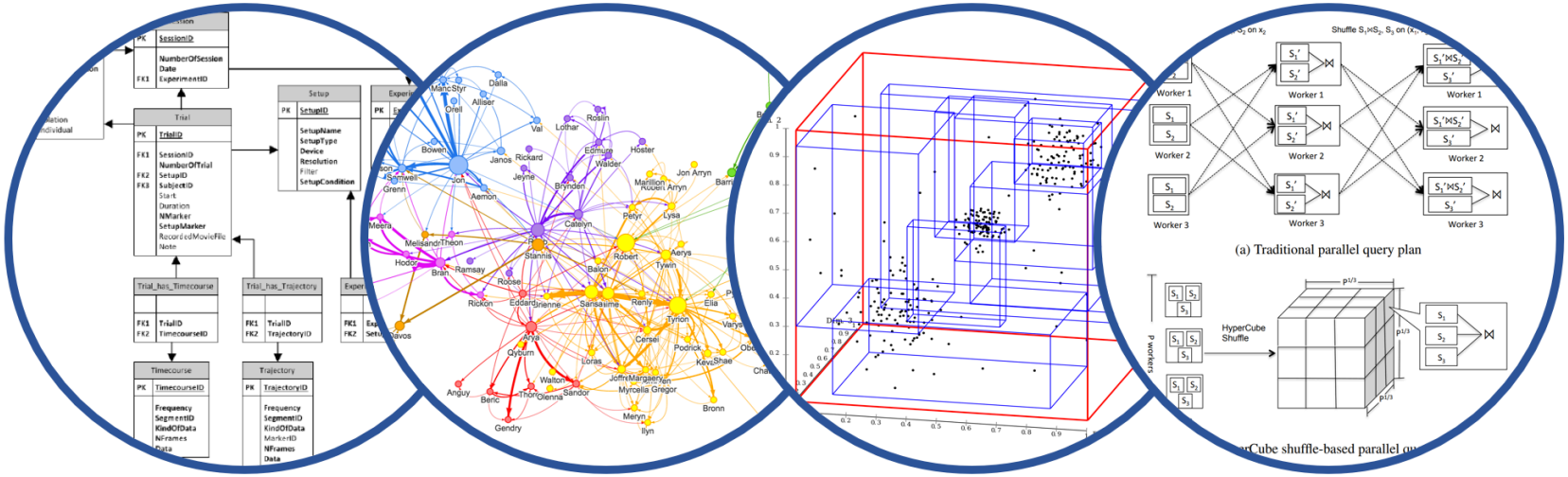


Database System Internals

Query Optimization (part 2)

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

Announcements

- HW 3 will be released tonight
- Lab 1 grades and feedback on Thursday
- Quiz 1+2 on Feb. 10th
  - Not as long as a midterm
  - Example posted on webpage calendar entry

Key Decisions for Implementation

Search Space

Optimization rules

Which algebraic laws do we apply?

Optimization algorithm

Optimization Rules – RA equivalencies

- Selections
  - Commutative: $\sigma_{c1}(\sigma_{c2}(R))$ same as $\sigma_{c2}(\sigma_{c1}(R))$
  - Cascading: $\sigma_{c1 \wedge c2}(R)$ same as $\sigma_{c2}(\sigma_{c1}(R))$

- Projections
  - Cascading

- Joins
  - Commutative : $R \bowtie S$ same as $S \bowtie R$
  - Associative: $R \bowtie (S \bowtie T)$ same as $(R \bowtie S) \bowtie T$

Example: Simple Algebraic Laws

- Example: R(A, B, C, D), S(E, F, G)

$\sigma_{F=3}(R \bowtie_{D=E} S) =$

$\sigma_{A=5 \text{ AND } G=9}(R \bowtie_{D=E} S) =$

Example: Simple Algebraic Laws

- Example: R(A, B, C, D), S(E, F, G)

$\sigma_{F=3}(R \bowtie_{D=E} S) = R \bowtie_{D=E} \sigma_{F=3}(S)$

$\sigma_{A=5 \text{ AND } G=9}(R \bowtie_{D=E} S) =$

Example: Simple Algebraic Laws

- Example: R(A, B, C, D), S(E, F, G)

$$\sigma_{F=3}(R \bowtie_{D=E} S) = R \bowtie_{D=E} \sigma_{F=3}(S)$$

$$\sigma_{A=5 \text{ AND } G=9}(R \bowtie_{D=E} S) = \sigma_{A=5}(R) \bowtie_{D=E} \sigma_{G=9}(S)$$

Commutativity, Associativity, Distributivity

$$R \cup S = S \cup R, \quad R \cup (S \cup T) = (R \cup S) \cup T$$
$$R \bowtie S = S \bowtie R, \quad R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$$

$$R \bowtie (S \cup T) = (R \bowtie S) \cup (R \bowtie T)$$

Laws Involving Selection

$$\sigma_{C \text{ AND } C'}(R) = \sigma_C(\sigma_{C'}(R)) = \sigma_C(R) \cap \sigma_{C'}(R)$$
$$\sigma_{C \text{ OR } C'}(R) = \sigma_C(R) \cup \sigma_{C'}(R)$$
$$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$$

$$\sigma_C(R - S) = \sigma_C(R) - S$$
$$\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$$
$$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$$

Assuming C on attributes of R

Laws Involving Projections

$$\Pi_M(R \bowtie S) = \Pi_M(\Pi_P(R) \bowtie \Pi_Q(S))$$

$$\Pi_M(\Pi_N(R)) = \Pi_M(R)$$

/* note that $M \subseteq N$ */

- Example R(A,B,C,D), S(E, F, G)

$$\Pi_{A,B,G}(R \bowtie_{D=E} S) = \Pi_{?}(\Pi_{?}(R) \bowtie_{D=E} \Pi_{?}(S))$$

Laws Involving Projections

$$\Pi_M(R \bowtie S) = \Pi_M(\Pi_P(R) \bowtie \Pi_Q(S))$$

$$\Pi_M(\Pi_N(R)) = \Pi_M(R)$$

/* note that $M \subseteq N$ */

- Example R(A,B,C,D), S(E, F, G)

$$\Pi_{A,B,G}(R \bowtie_{D=E} S) = \Pi_{A,B,G}(\Pi_{A,B,D}(R) \bowtie_{D=E} \Pi_{E,G}(S))$$

Laws for grouping and aggregation

$$\gamma_{A,\,agg(D)}(R(A,B) \bowtie_{B=C} S(C,D)) =$$
$$\gamma_{A,\,agg(D)}(R(A,B) \bowtie_{B=C} (\gamma_{C,\,agg(D)} S(C,D)))$$

Laws for grouping and aggregation

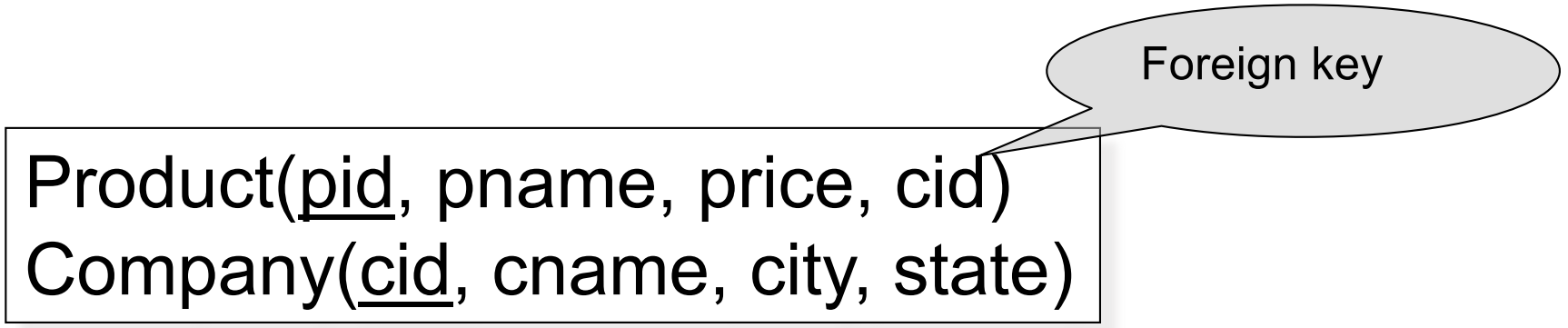
$$\delta(\gamma_{A,\,agg(B)}(R)) = \gamma_{A,\,agg(B)}(R)$$

$$\gamma_{A,\,agg(B)}(\delta(R)) = \gamma_{A,\,agg(B)}(R)$$

if agg is "duplicate insensitive"

Which of the following are "duplicate insensitive" ?
sum, count, avg, min, max

Laws Involving Constraints



$$\Pi_{\text{pid, price}}(\text{Product} \bowtie_{\text{cid=cid}} \text{Company}) = \Pi_{\text{pid, price}}(\text{Product})$$

Search Space Challenges

- Search space is huge!
  - Many possible equivalent trees
  - Many implementations for each operator
  - Many access paths for each relation
    - File scan or index + matching selection condition
- Cannot consider ALL plans
  - Heuristics: only partial plans with “low” cost

Key Decisions

Search Space

Optimization rules

Optimization algorithm

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

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

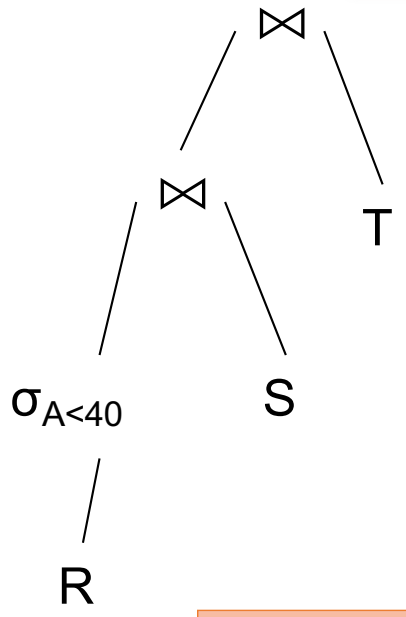
The Three Parts of an Optimizer

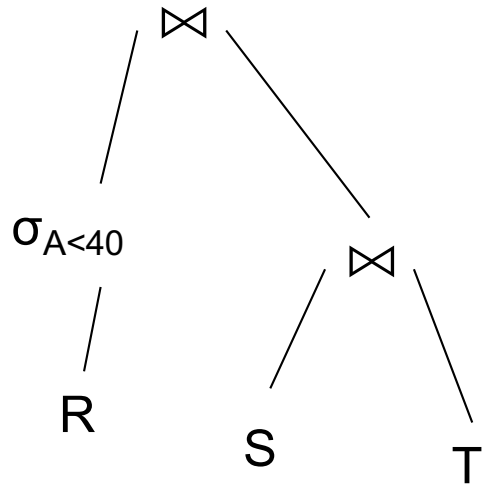
- Cost estimation
  - Based on cardinality estimation
- Search space
- Search algorithm

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
```





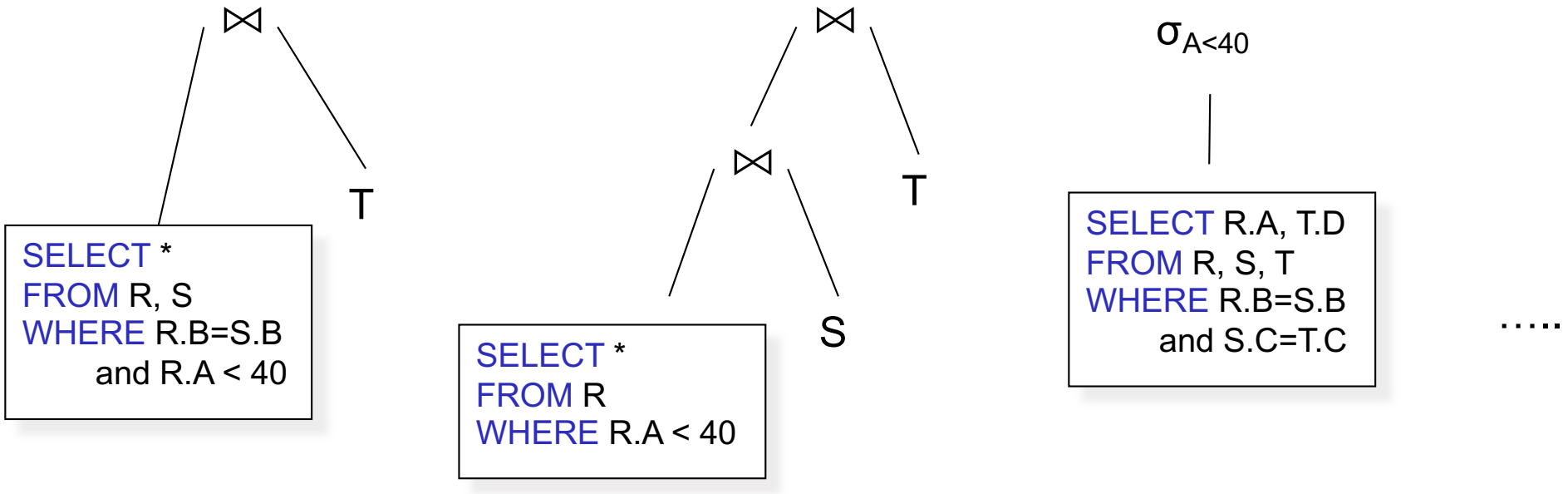
Why is this search space inefficient ?

Answer: No way to do early pruning

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
```

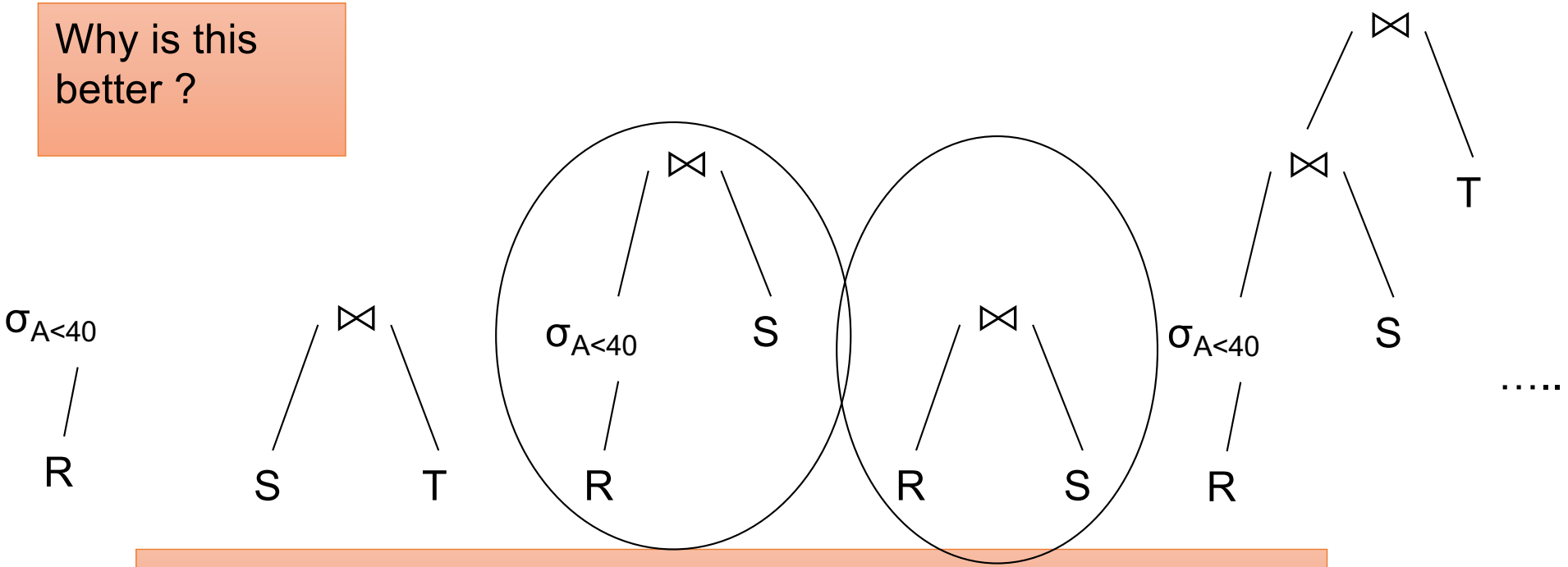


Bottom-up Partial Plans

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

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

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

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

Dynamic Programming

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

- 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

Dynamic Programming

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

- **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

Dynamic Programming

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

- **Step 2**: For each Q ⊆{R_1, …, 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

Dynamic Programming

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

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

Example

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

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

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

- Every join selectivity is 0.001

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			

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		

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		

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		

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	Clustered index scan U.F	100
RS	10000		
RT	6000		
RU	2000		
ST	15000		
SU	5000		
TU	3000		
RST	30000		
RSU	10000		
RTU	6000		
STU	15000		
RSTU	30000		

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	Clustered index scan U.F	100
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		

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	Clustered index scan U.F	100
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		

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	Clustered index scan U.F	100
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		

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	Clustered index scan U.F	100
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		

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	Clustered index scan U.F	100
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	…

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

Discussion

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

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?

Discussion

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

Given a query with n relations R1, …, Rn

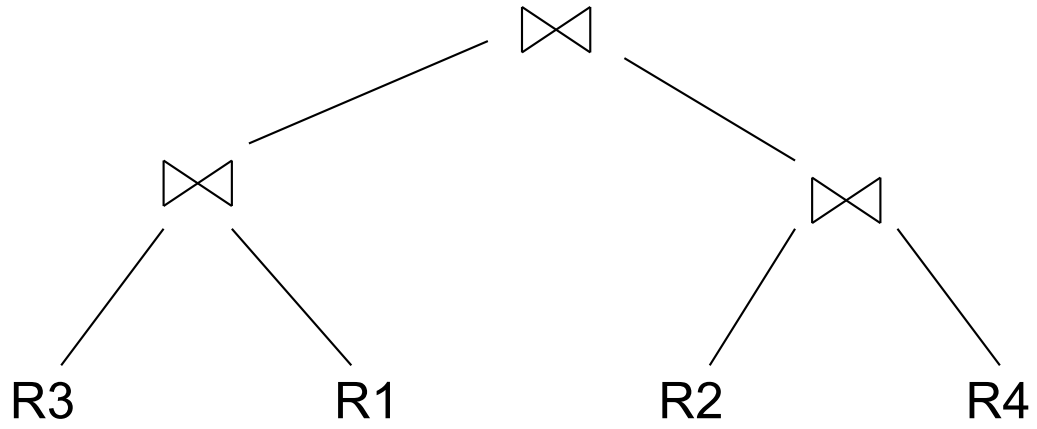
- How many entries do we have in the dynamic programming table?
  - A: $2^n - 1$
- For each entry, how many alternative plans do we need to inspect?
  - A: for each entry with k tables, examine $2^k - 2$ plans

Reducing the Search Space

- Left-linear trees
- No cartesian products

Join Trees

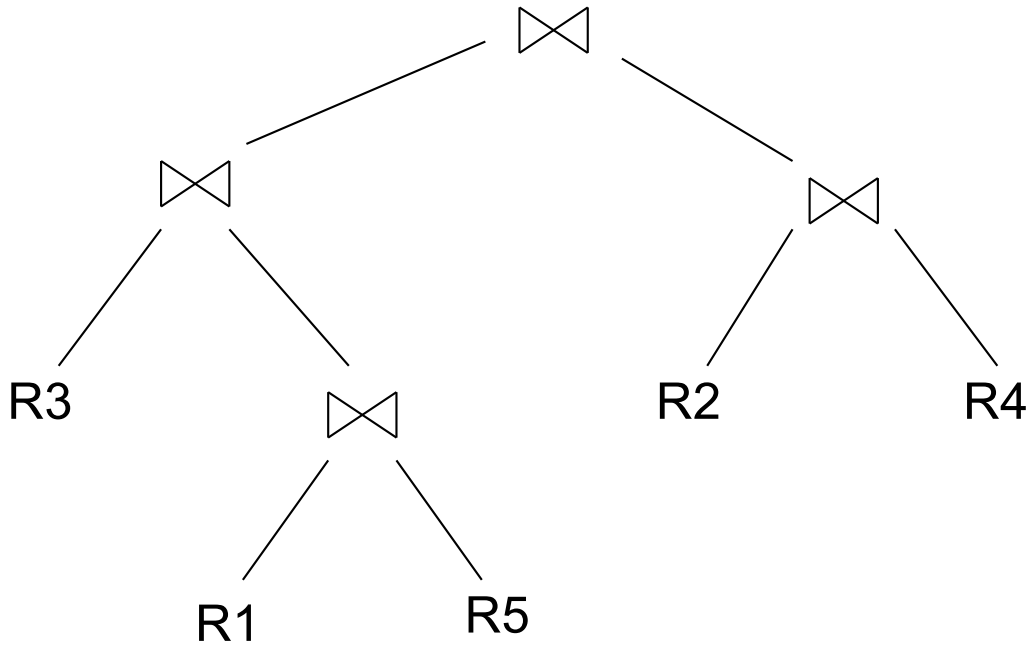
- R1 ⋈ R2 ⋈ …. ⋈ Rn
- Join tree:



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

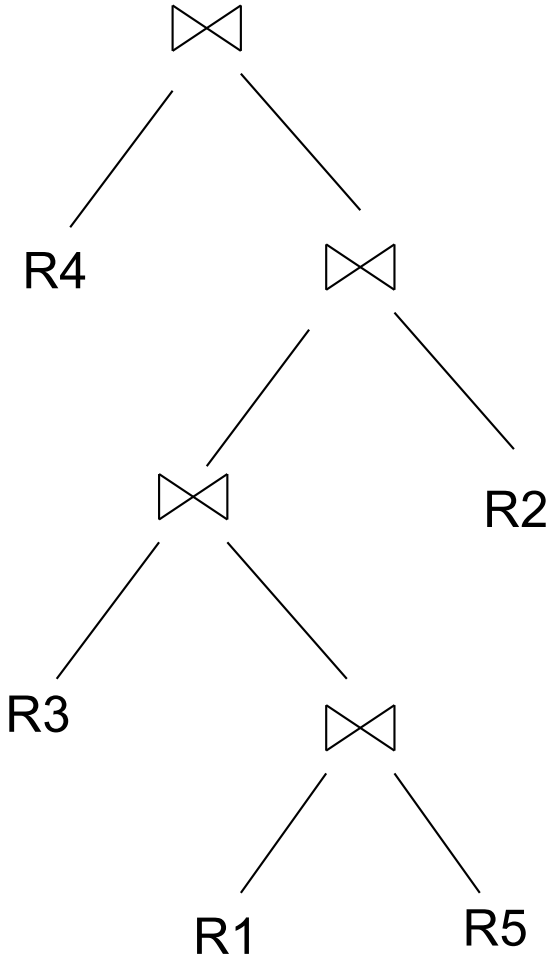
Types of Join Trees

- Bushy:



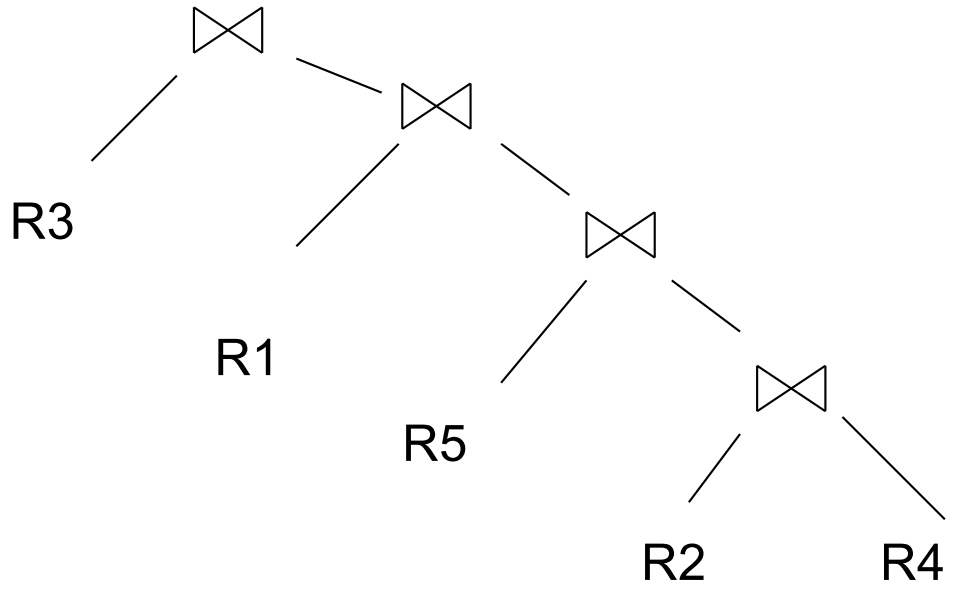
Types of Join Trees

- Linear :



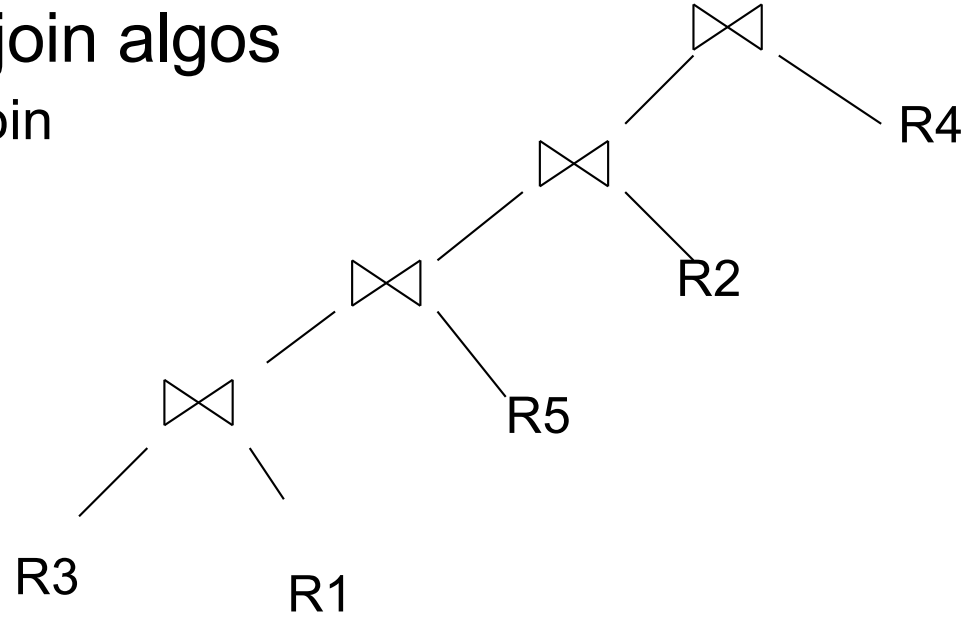
Types of Join Trees

- Right deep:



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

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