## Distributed Databases

CS347
Lecture 14
May 30, 2001

1

## Topics for the Day

- Query processing in distributed databases
  - Localization
  - Distributed query operators
  - Cost-based optimization

2

## Query Processing Steps

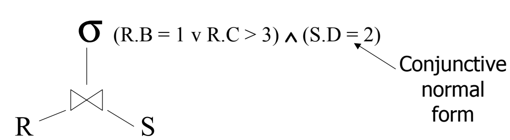
- Decomposition
  - Given SQL query, generate one or more algebraic query trees
- Localization
  - Rewrite query trees, replacing relations by fragments
- Optimization
  - Given cost model + one or more localized query trees
  - Produce minimum cost query execution plan

3

## Decomposition

- Same as in a centralized DBMS
- Normalization (usually into relational algebra)

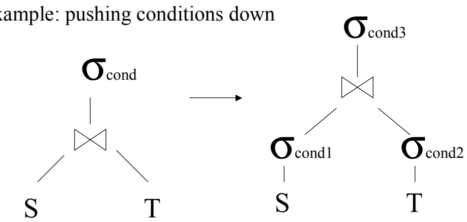
```
Select A,C
From R Natural Join S
Where (R.B = 1 and S.D = 2) or (R.C > 3 and S.D = 2)
```




4

## Decomposition

- Redundancy elimination
  - (S.A = 1) ∧ (S.A > 5) ⇒ False
  - (S.A < 10) ∧ (S.A < 5) ⇒ S.A < 5
- Algebraic Rewriting
  - Example: pushing conditions down




5

## Localization Steps

1. Start with query tree
2. Replace relations by fragments
3. Push ∪ up & π,σ down (CS245 rules)
4. Simplify – eliminating unnecessary operations

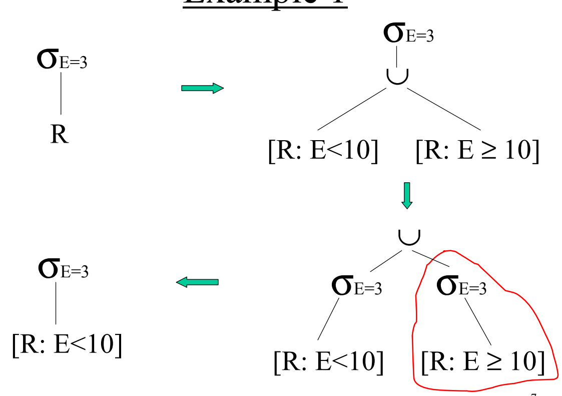
Note: To denote fragments in query trees

[R: cond]

Relation that fragment belongs to — Condition its tuples satisfy

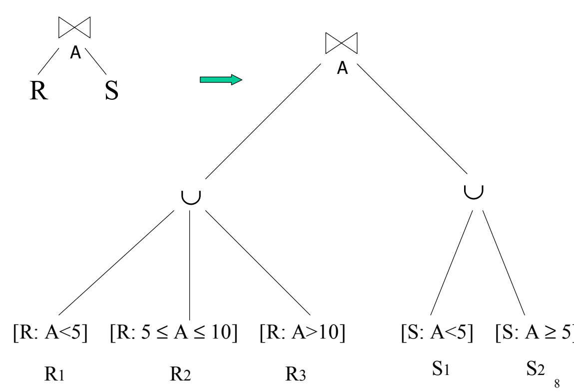
6

## Example 1

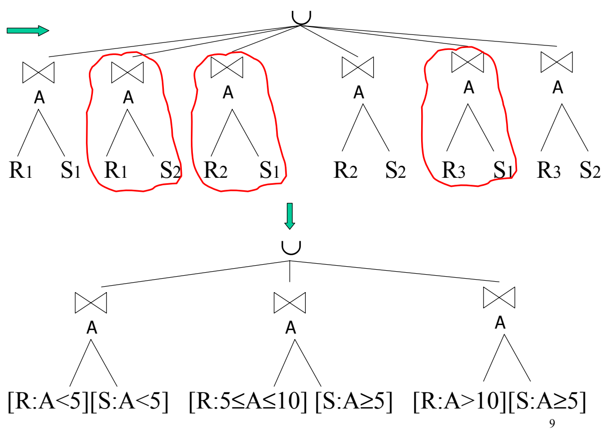



7

## Example 2




8



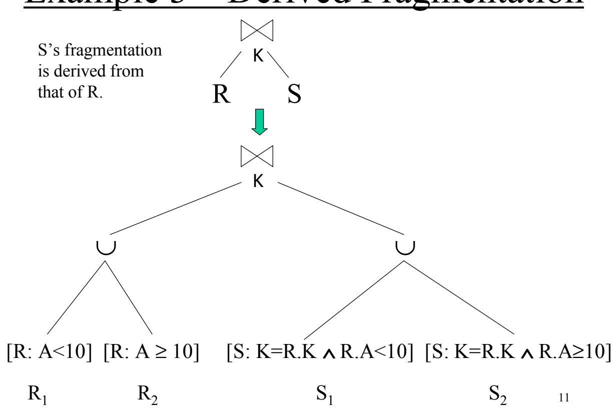

# Rules for Horiz. Fragmentation

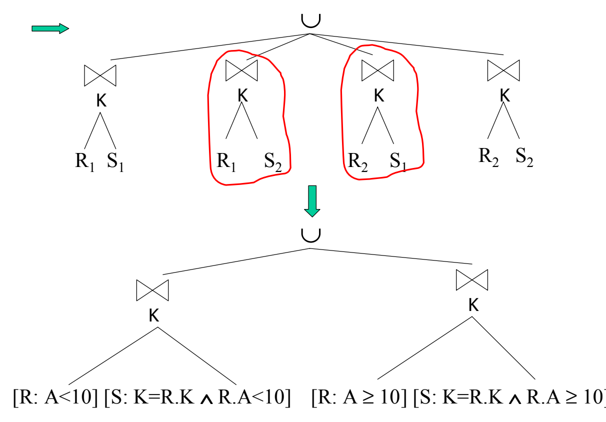
- $\sigma_{C1}$[R: $C_2$] $\Rightarrow$ [R: $C_1 \wedge C_2$]
- [R: False] $\Rightarrow$ Ø
- [R: $C_1$] $\bowtie_A$ [S: $C_2$] $\Rightarrow$ [R $\bowtie_A$ S: $C_1 \wedge C_2 \wedge$ R.A = S.A]
- In Example 1:
  $\sigma_{E=3}$[$R_2$: E ≥ 10] $\Rightarrow$ [$R_2$: E=3 $\wedge$ E ≥ 10]
  $\Rightarrow$ [$R_2$: False] $\Rightarrow$ Ø
- In Example 2:
  [R: A<5] $\bowtie_A$ [S: A ≥ 5]
  $\Rightarrow$ [R $\bowtie_A$ S: R.A < 5 $\wedge$ S.A ≥ 5 $\wedge$ R.A = S.A]
  $\Rightarrow$ [R $\bowtie_A$ S: False] $\Rightarrow$ Ø

10

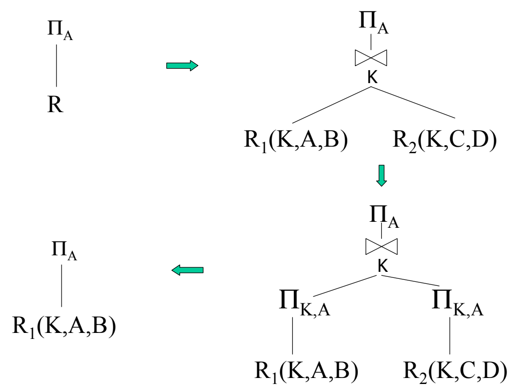
# Example 3 – Derived Fragmentation

S's fragmentation is derived from that of R.



## Example 4 – Vertical Fragmentation




13

## Rule for Vertical Fragmentation

- Given vertical fragmentation of R(A):

  $R_i = \Pi_{Ai}(R), \; A_i \subseteq A$
- For any $B \subseteq A$:

  $\Pi_B(R) = \Pi_B [ \bowtie_i R_i \mid B \cap A_i \neq \emptyset ]$

14

## Parallel/Distributed Query Operations

- Sort
  - Basic sort
  - Range-partitioning sort
  - Parallel external sort-merge
- Join
  - Partitioned join
  - Asymmetric fragment and replicate join
  - General fragment and replicate join
  - Semi-join programs
- Aggregation and duplicate removal

15

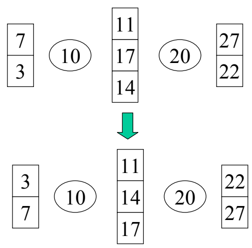
## Parallel/distributed sort

- <u>Input:</u> relation R on
  - single site/disk
  - fragmented/partitioned by sort attribute
  - fragmented/partitioned by some other attribute
- <u>Output:</u> sorted relation R
  - single site/disk
  - individual sorted fragments/partitions

16

# Basic sort

- Given R(A,…) range partitioned on attribute A, sort R on A

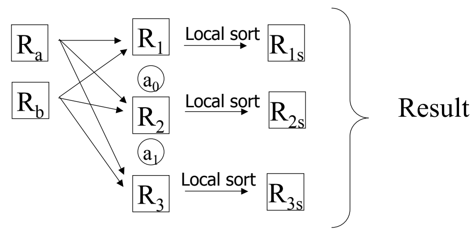



- Each fragment is sorted independently
- Results shipped elsewhere if necessary

17

# Range partitioning sort

- Given R(A,….) located at one or more sites, <u>not</u> fragmented on A, sort R on A
- <u>Algorithm:</u> range partition on A and then do basic sort




18

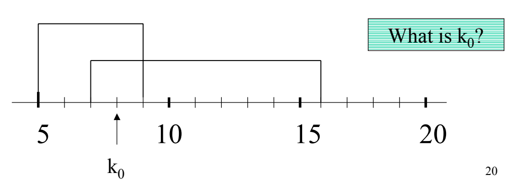
# Selecting a partitioning vector

- Possible centralized approach using a "coordinator"
  - Each site sends *statistics* about its fragment to coordinator
  - Coordinator decides # of sites to use for local sort
  - Coordinator computes and distributes partitioning vector
- For example,
  - Statistics could be (min sort key, max sort key, # of tuples)
  - Coordinator tries to choose vector that equally partitions relation

19

# Example

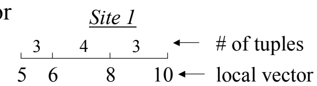
- Coordinator receives:
  - From site 1: Min 5, Max 9, 10 tuples
  - From site 2: Min 7, Max 16, 10 tuples
- Assume sort keys distributed uniformly within [min,max] in each fragment
- Partition R into two fragments




20

# Variations

- Different kinds of statistics
  - Local partitioning vector
  - Histogram

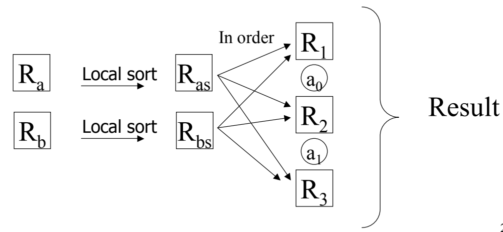



- Multiple rounds between coordinator and sites
  - Sites send statistics
  - Coordinator computes and distributes initial vector V
  - Sites tell coordinator the number of tuples that fall in each range of V
  - Coordinator computes final partitioning vector $V_f$



# Parallel external sort-merge

- Local sort
- Compute partition vector
- Merge sorted streams at final sites





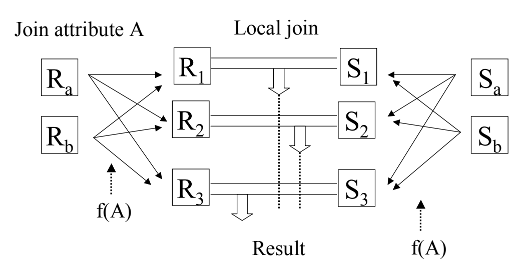

# Parallel/distributed join

Input: Relations R, S
May or may not be partitioned

Output: R ⋈ S
Result at one or more sites



# Partitioned Join



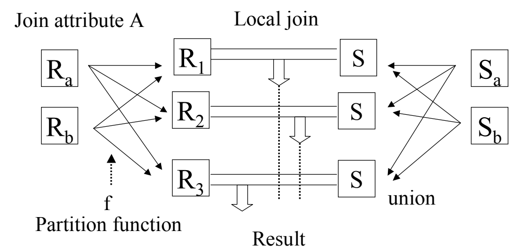

Note: Works only for equi-joins

# Partitioned Join

- Same partition function (f) for both relations
- f can be range or hash partitioning
- Any type of local join (nested-loop, hash, merge, etc.) can be used
- Several possible scheduling options. Example:
  - partition R; partition S; join
  - partition R; build local hash table for R; partition S and join
- Good partition function important
  - Distribute join load evenly among sites



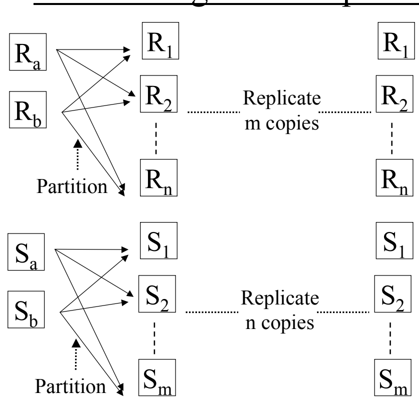
# Asymmetric fragment + replicate join



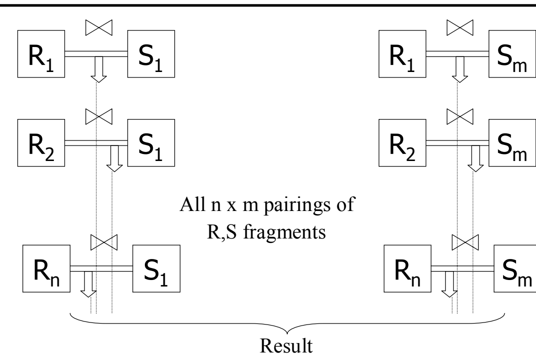

- Any partition function f can be used (even round-robin)
- Can be used for any kind of join, not just equi-joins



# General fragment + replicate join






- Asymmetric F+R join is a special case of general F+R.
- Asymmetric F+R is useful when S is small.

# Semi-join programs

- Used to reduce communication traffic during join processing
- $R \bowtie S = (R \ltimes S) \bowtie S$

  $= R \bowtie (S \ltimes R)$

  $= (R \ltimes S) \bowtie (S \ltimes R)$

29

# Example

S

| A | B |
|---|---|
| 2 | a |
| 10 | b |
| 25 | c |
| 30 | d |

R

| A | C |
|---|---|
| 3 | x |
| 10 | y |
| 15 | z |
| 25 | w |
| 32 | x |

$\Pi_A(S) = [2,10,25,30]$

Compute $S \bowtie (R \ltimes S)$

$R \ltimes S =$

| | |
|---|---|
| 10 | y |
| 25 | w |

- Using semi-join, communication cost = 4 A + 2 (A + C) + result
- Directly joining R and S, communication cost = 4 (A + B) + result

30

# Comparing communication costs

- Say R is the smaller of the two relations R and S
- $(R \ltimes S) \bowtie S$ is cheaper than $R \bowtie S$ if

  size $(\Pi_A S)$ + size $(R \ltimes S)$ < size (R)
- Similar comparisons for other types of semi-joins
- Common implementation trick:
  - Encode $\Pi_A S$ (or $\Pi_A R$) as a bit vector
  - 1 bit per domain of attribute A

0 0 1 1 0 1 0 0 0 0 1 0 1 0 0

31

# n-way joins

- To compute $R \bowtie S \bowtie T$
  - Semi-join program 1: $R' \bowtie S' \bowtie T$

    where $R' = R \ltimes S$ & $S' = S \ltimes T$
  - Semi-join program 2: $R'' \bowtie S' \bowtie T$

    where $R'' = R \ltimes S'$ & $S' = S \ltimes T$
  - Several other options

- In general, number of options is exponential in the number of relations

32

# Other operations

- Duplicate elimination
  - Sort first (in parallel), then eliminate duplicates in the result
  - Partition tuples (range or hash) and eliminate duplicates locally
- Aggregates
  - Partition by grouping attributes; compute aggregates locally at each site



# Example

$R_a$

| # | dept | sal |
|---|---|---|
| 1 | toy | 10 |
| 2 | toy | 20 |
| 3 | sales | 15 |

$R_b$

| # | dept | sal |
|---|---|---|
| 4 | sales | 5 |
| 5 | toy | 20 |
| 6 | mgmt | 15 |
| 7 | sales | 10 |
| 8 | mgmt | 30 |

→

| # | dept | sal |
|---|---|---|
| 1 | toy | 10 |
| 2 | toy | 20 |
| 5 | toy | 20 |
| 6 | mgmt | 15 |
| 8 | mgmt | 30 |

sum →

| dept | sum |
|---|---|
| toy | 50 |
| mgmt | 45 |

→

| # | dept | sal |
|---|---|---|
| 3 | sales | 15 |
| 4 | sales | 5 |
| 7 | sales | 10 |

sum →

| dept | sum |
|---|---|
| sales | 30 |

sum(sal) group by dept



# Example

$R_a$

| # | dept | sal |
|---|---|---|
| 1 | toy | 10 |
| 2 | toy | 20 |
| 3 | sales | 15 |

$R_b$

| # | dept | sal |
|---|---|---|
| 4 | sales | 5 |
| 5 | toy | 20 |
| 6 | mgmt | 15 |
| 7 | sales | 10 |
| 8 | mgmt | 30 |

sum →

| dept | sum |
|---|---|
| toy | 30 |
| toy | 20 |
| mgmt | 45 |

sum →

| dept | sum |
|---|---|
| toy | 50 |
| mgmt | 45 |

sum →

| dept | sum |
|---|---|
| sales | 15 |
| sales | 15 |

sum →

| dept | sum |
|---|---|
| sales | 30 |

Does this work for all kinds of aggregates?

Aggregate during partitioning to reduce communication cost

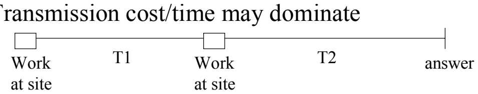


# Query Optimization

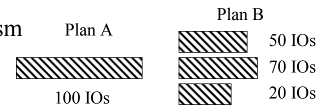
- Generate query execution plans (QEPs)
- Estimate cost of each QEP ($,time,…)
- Choose minimum cost QEP
- What's different for distributed DB?
  - New strategies for some operations (semi-join, range-partitioning sort,…)
  - Many ways to assign and schedule processors
  - Some factors besides number of IO's in the cost model

# Cost estimation

- In centralized systems - estimate sizes of intermediate relations
- For distributed systems
  – Transmission cost/time may dominate




  – Account for parallelism




  – Data distribution and result re-assembly cost/time

37

# Optimization in distributed DBs

- Two levels of optimization
- Global optimization
  – Given localized query and cost function
  – Output optimized (min. cost) QEP that includes relational and communication operations on fragments
- Local optimization
  – At each site involved in query execution
  – Portion of the QEP at a given site optimized using techniques from centralized DB systems

38

# Search strategies

1. Exhaustive (with pruning)
2. Hill climbing (greedy)
3. Query separation

39

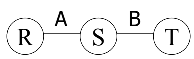
## Exhaustive with Pruning

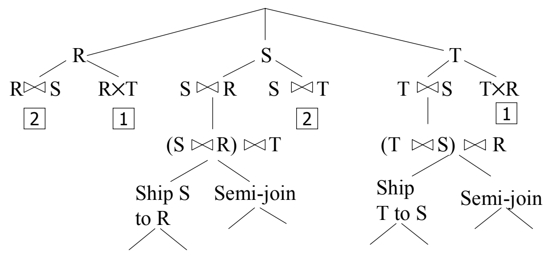
- A fixed set of techniques for each relational operator
- Search space = “all” possible QEPs with this set of techniques
- Prune search space using heuristics
- Choose minimum cost QEP from rest of search space

40

## Example

|R|>|S|>|T|



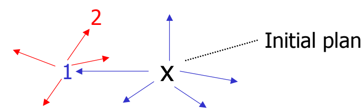




[1] Prune because cross-product not necessary

[2] Prune because larger relation first



## Hill Climbing

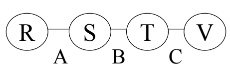



- Begin with initial feasible QEP
- At each step, generate a set S of new QEPs by applying ‘transformations’ to current QEP
- Evaluate cost of each QEP in S
- Stop if no improvement is possible
- Otherwise, replace current QEP by the minimum cost QEP from S and iterate



## Example

R⋈S⋈T⋈V




| Rel. | Site | # of tuples |
|---|---|---|
| R | 1 | 10 |
| S | 2 | 20 |
| T | 3 | 30 |
| V | 4 | 40 |

- Goal: minimize communication cost
- Initial plan: send all relations to one site
  - To site 1: cost=20+30+40= 90
  - To site 2: cost=10+30+40= 80
  - To site 3: cost=10+20+40= 70
  - To site 4: cost=10+20+30= (60)
- Transformation: send a relation to its neighbor



## Local search

- Initial feasible plan

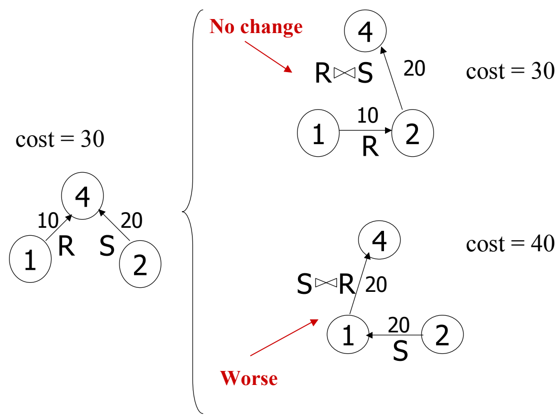
  P0: R (1 → 4); S (2 → 4); T (3 → 4)

  Compute join at site 4

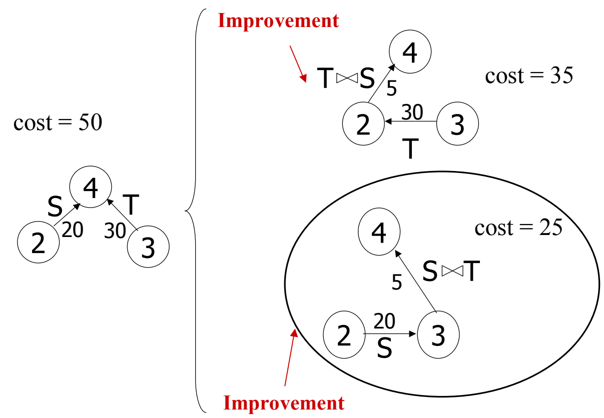
- Assume following sizes: R ⋈ S ⇒ 20

  S ⋈ T ⇒ 5

  T ⋈ V ⇒ 1



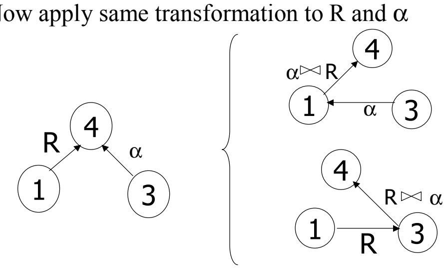

45




46

## Next iteration

- P1: S (2 → 3); R (1 → 4); α (3 → 4)
  where α = S⋈T
  Compute answer at site 4
- Now apply same transformation to R and α




47

## Resources

- Ozsu and Valduriez. “Principles of Distributed Database Systems” – Chapters 7, 8, and 9.

48