

CS 347:  
Distributed Databases and  
Transaction Processing  
**Notes03: Query Processing**

Hector Garcia-Molina  
Zoltan Gyongyi

CS 347

Notes 03

1

Query Processing

- Decomposition
- Localization
- Optimization

CS 347

Notes 03

2

Decomposition

- Same as in centralized system
- Normalization
- Eliminating redundancy
- Algebraic rewriting

CS 347

Notes 03

3

Normalization

- Convert from general language to a "standard" form (e.g., relational algebra)

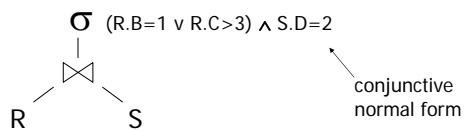
CS 347

Notes 03

4

Example

select A, C  
from R, S  
where (R.B=1 and S.D=2) or (R.C>3 and S.D=2)



CS 347

Notes 03

5

Also: Detect invalid expressions

E.g.: select \* from R where R.A=3  
☒ R does not have "A" attribute

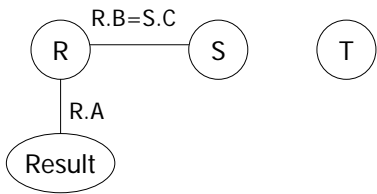
E.g.: select R.A  
from R, S, T  
where R.B=S.C

CS 347

Notes 03

6

Note: Query graph useful for detecting last problem

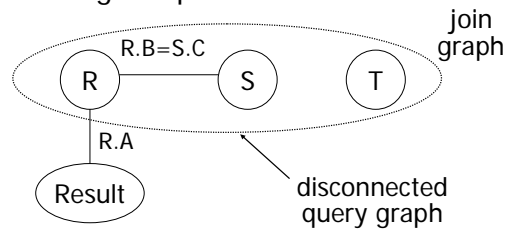


CS 347

Notes 03

7

Note: Query graph useful for detecting last problem



CS 347

Notes 03

8

Eliminate redundancy

E.g.: in conditions

$$(S.A=1) \wedge (S.A>5) \Rightarrow \text{False}$$

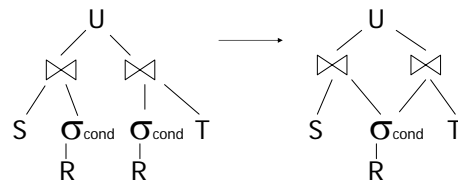
$$(S.A<10) \wedge (S.A<5) \Rightarrow S.A<5$$

CS 347

Notes 03

9

E.g.: common sub-expressions



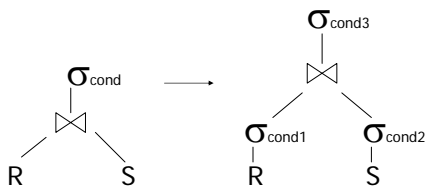
CS 347

Notes 03

10

Algebraic rewriting

E.g.: push conditions down



CS 347

Notes 03

11

- After decomposition:
  - One or more algebraic query trees on relations
- Localization:
  - Replace relations by corresponding fragments

CS 347

Notes 03

12

### Localization steps

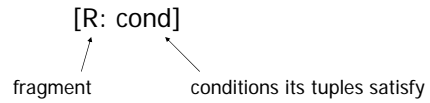
- (1) Start with query
- (2) Replace relations by fragments
- (3) Push  $\begin{cases} \cup \text{ up} \\ \pi, \sigma \text{ down} \end{cases}$  (use CS 245 rules)
- (4) Simplify – eliminate unnecessary operations

CS 347

Notes 03

13

### Notation for fragment



CS 347

Notes 03

14

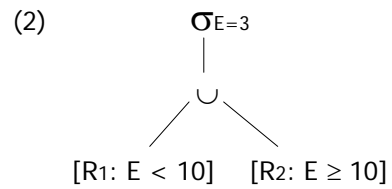
### Example A



CS 347

Notes 03

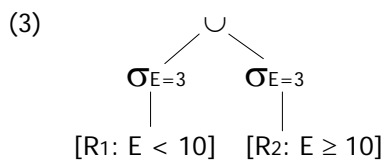
15



CS 347

Notes 03

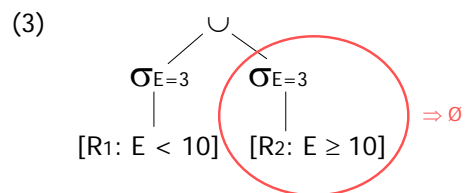
16



CS 347

Notes 03

17



CS 347

Notes 03

18

(4)

$$\begin{array}{c} \sigma_{E=3} \\ | \\ [R1: E < 10] \end{array}$$

CS 347

Notes 03

19

### Rule 1

- (A)  $\sigma_{c1}[R: c2] \Rightarrow \sigma_{c1}[R: c1 \wedge c2]$
- (B)  $[R: \text{False}] \Rightarrow \emptyset$

CS 347

Notes 03

20

### In example A:

$$\begin{aligned} \sigma_{E=3}[R2: E \geq 10] &\Rightarrow [\sigma_{E=3} R2: E=3 \wedge E \geq 10] \\ &\Rightarrow [\sigma_{E=3} R2: \text{False}] \\ &\Rightarrow \emptyset \end{aligned}$$

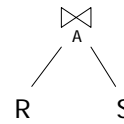
CS 347

Notes 03

21

### Example B

(1)



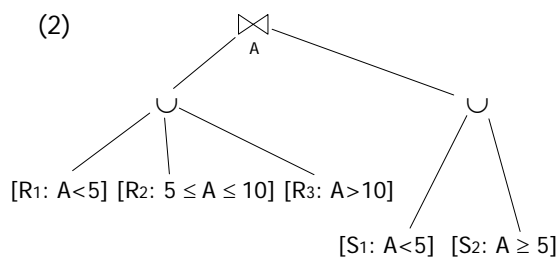
A = common attribute

CS 347

Notes 03

22

(2)

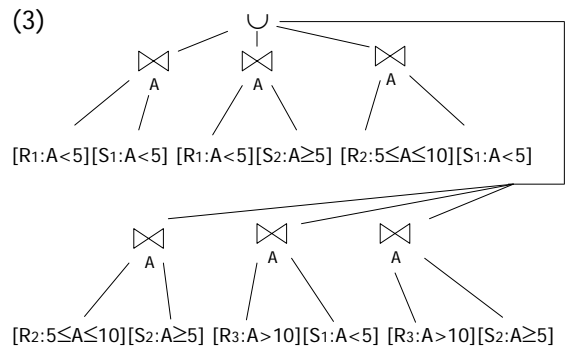


CS 347

Notes 03

23

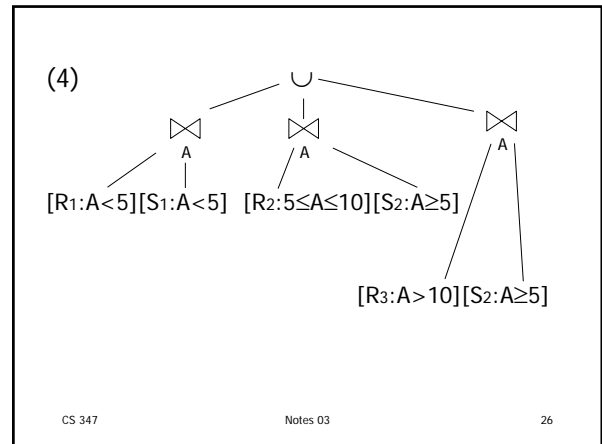
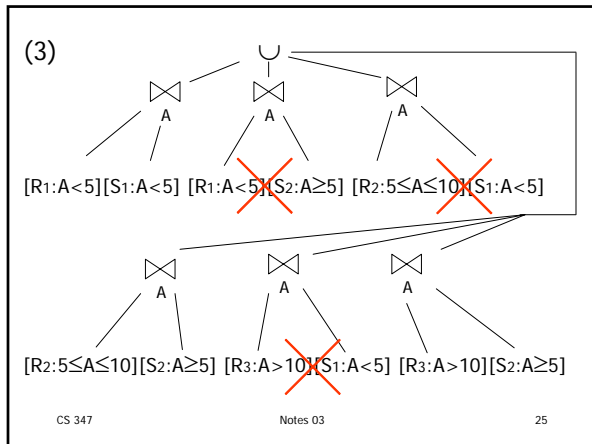
(3)



CS 347

Notes 03

24



Rule 2

$$[R: C_1] \underset{A}{\bowtie} [S: C_2] \Rightarrow$$

$$[R \underset{A}{\bowtie} S: C_1 \wedge C_2 \wedge R.A = S.A]$$

CS 347 Notes 03 27

In step 4 of Example B:

$$[R1: A<5] \underset{A}{\bowtie} [S2: A \geq 5]$$

$$\Rightarrow [R1 \underset{A}{\bowtie} S2: R1.A < 5 \wedge S2.A \geq 5 \wedge R1.A = S2.A]$$

$$\Rightarrow [R1 \underset{A}{\bowtie} S2: \text{False}] \Rightarrow \emptyset$$

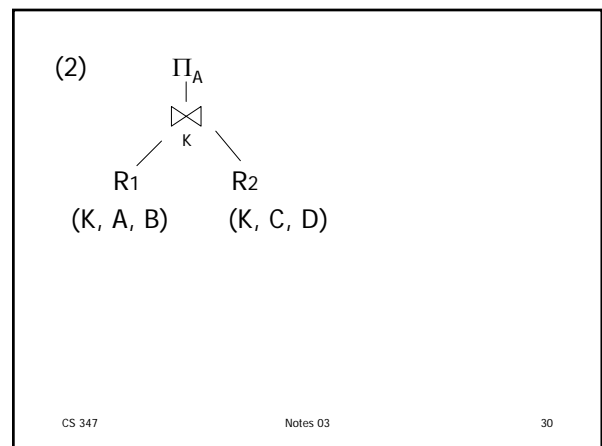
CS 347 Notes 03 28

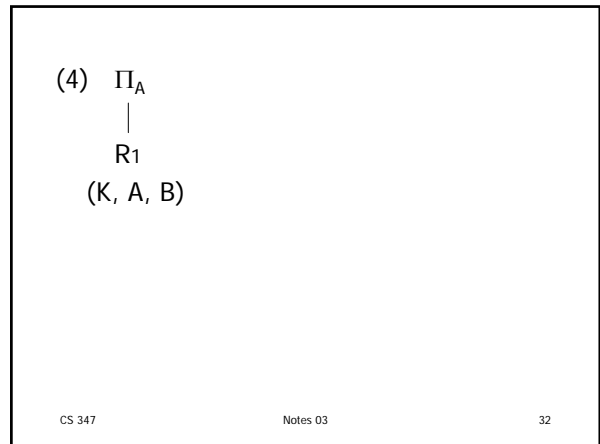
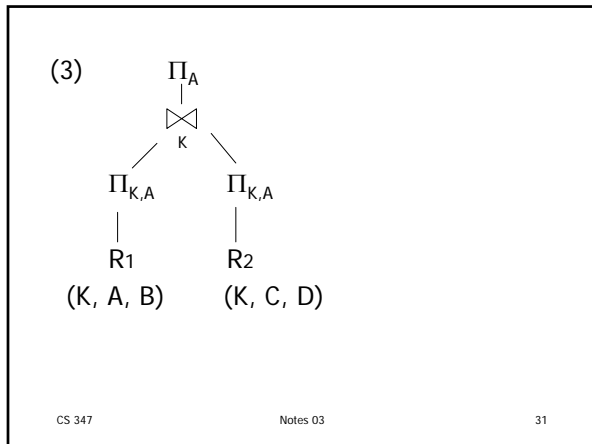
• Localization with vertical fragmentation

Example C

(1)  $\Pi_A$   
 $\underset{R}{\downarrow}$   $\left\{ \begin{array}{l} R1(K, A, B) \\ R2(K, C, D) \end{array} \right.$

CS 347 Notes 03 29





Rule 3

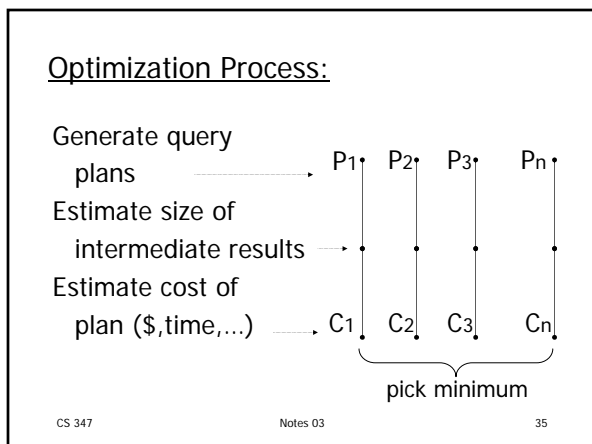
- Given vertical fragmentation of R:  
 $R_i = \Pi_{A_i}(R), A_i \subseteq A$
- Then for any  $B \subseteq A$ :  
 $\Pi_B(R) = \Pi_B \left[ \bigotimes_i R_i \mid B \cap A_i \neq \emptyset \right]$

CS 347 Notes 03 33

Summary - Query Processing

- Decomposition ✓
- Localization ✓
- Optimization
  - Overview
  - Tricks for joins + other operations
  - Strategies for optimization

CS 347 Notes 03 34



Differences from centralized optimization:

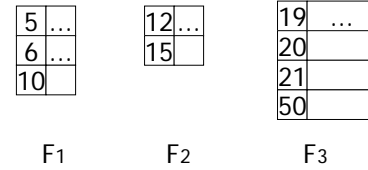
- New strategies for some operations (semi-join, range-partitioning, sort, ...)
- Many ways to assign and schedule processors

CS 347 Notes 03 36

Parallel/distributed sort

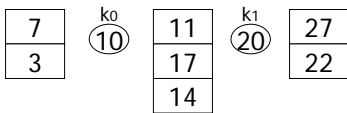
- Input:
- (a) relation R on single site/disk
  - (b) R fragmented/partitioned by sort attribute
  - (c) R fragmented/partitioned by other attribute

- Output:
- (a) sorted R on single site/disk
  - (b) fragments/partitions sorted



Basic sort

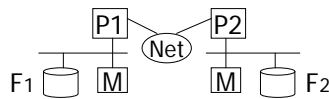
- $R(K, \dots)$ ; sort on K
- Fragmented (range partitioned) on K  
Vector: [  $k_0, k_1, \dots, k_n$  ]



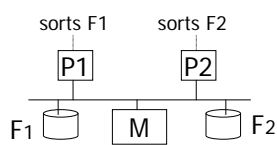
- Algorithm:
  - Sort each fragment independently
  - If necessary, ship results

⇒ Same idea on different architectures:

Shared nothing

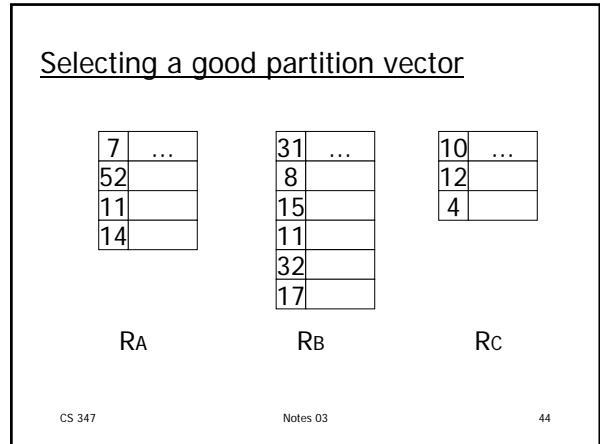
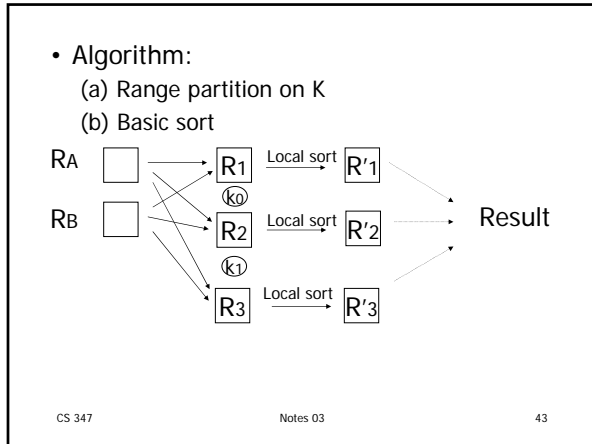


Shared memory



Range partitioning sort

- $R(K, \dots)$ ; sort on K
- R located at one or more sites/disks not fragmented on K



Example

- Each site sends to coordinator:
  - Min sort key
  - Max sort key
  - Number of tuples
- Coordinator computes vector and distributes to sites (also decides # of sites for local sorts)

CS 347 Notes 03 45

Sample scenario

Coordinator receives:

SA: Min=5 Max=10 # = 10 tuples

SB: Min=7 Max=17 # = 10 tuples

CS 347 Notes 03 46

Sample scenario

Coordinator receives:

SA: Min=5 Max=10 # = 10 tuples

SB: Min=7 Max=17 # = 10 tuples

Expected tuples:

CS 347 Notes 03 47

Expected tuples:

Expected tuples with key <  $k_0$  =  $\frac{\text{Total tuples}}{2}$

$2(k_0 - 5) + (k_0 - 7) = 10$

$3 k_0 = 10 + 10 + 7 = 27$

$k_0 = 9$

CS 347 Notes 03 48

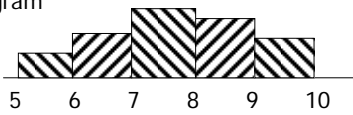
### Variations

- Send more info to coordinator

– Partition vector for local site

E.g., SA:  $\frac{3}{5} \frac{3}{6} \frac{3}{8} \frac{3}{10}$  ← # tuples  
 ← local vector

– Histogram



CS 347

Notes 03

49

⇔ More than one round

- E.g.:
- (1) Sites send range and # tuples
  - (2) Coordinator returns "preliminary" vector  $V_0$
  - (3) Sites tell coordinator how many tuples in each  $V_0$  range
  - (4) Coordinator computes final vector  $V_f$

CS 347

Notes 03

50

- Can you come up with a distributed algorithm? (no coordinator)

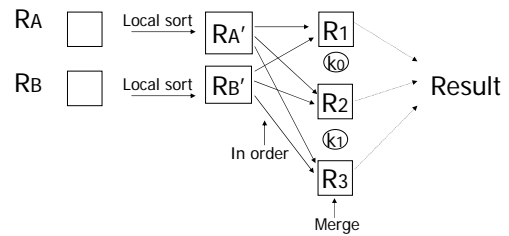
CS 347

Notes 03

51

### Parallel external sort-merge

- Same as range-partitioning sort, except sort first



CS 347

Notes 03

52

### Parallel/distributed join

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

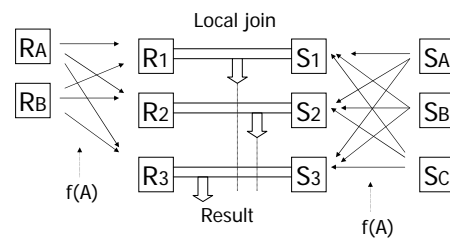
Output:  $R \bowtie S$   
 Result at one or more sites

CS 347

Notes 03

53

### Partitioned join (equi-join)



CS 347

Notes 03

54

### Notes

- Same partition function  $f$  is used for both  $R$  and  $S$  (applied to join attribute  $A$ )
- $f$  can be range or hash partitioning
- Local join can be of any type (use any CS 245 optimization)
- Various scheduling options, e.g.,
  - (a) partition  $R$ ; partition  $S$ ; join
  - (b) partition  $R$ ; build local hash table for  $R$ ; partition  $S$  and hash-join

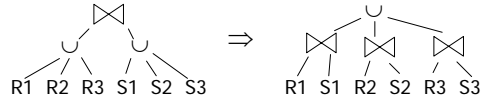
CS 347

Notes 03

55

### More notes

- We already know why partition-join works:



- Useful to give this type of join a name, because we may want to partition data to make partition-join possible (especially in a parallel DB system)

CS 347

Notes 03

56

### Even more notes

- Selecting a good partition function  $f$  is very important:
  - Number of fragments
  - Hash function
  - Partition vector

CS 347

Notes 03

57

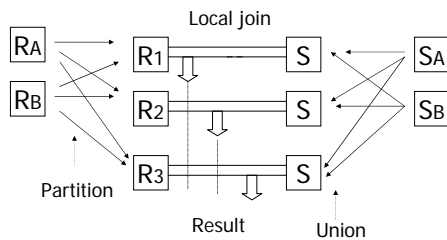
- Good partition vector
  - Goal:  $|R_i| + |S_i|$  the same
  - Can use coordinator to select

CS 347

Notes 03

58

### Asymmetric fragment + replicate join



CS 347

Notes 03

59

### Notes:

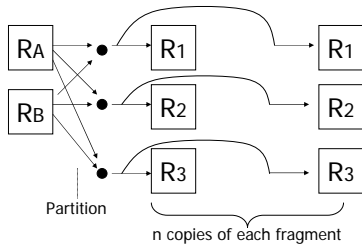
- Can use any partition function  $f$  for  $R$  (even round robin)
- Can do any join, not just equi-join  
e.g.:  $R \bowtie S$   
 $R.A < S.B$

CS 347

Notes 03

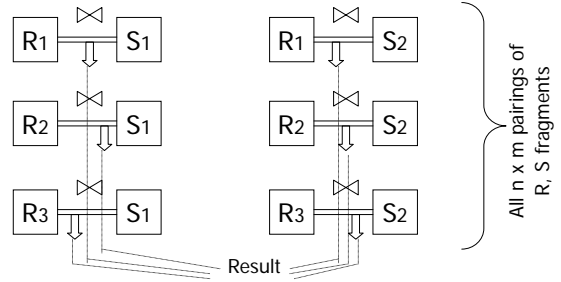
60

General fragment and replicate join



CS 347 Notes 03 61

• S is partitioned in similar fashion



CS 347 Notes 03 62

Notes

- Asymmetric F+R join is special case of general F+R
- Asymmetric F+R may be good if S small
- Works for non-equi-joins

CS 347 Notes 03 63

Semi-join

- Goal: reduce communication traffic
- $R \bowtie_A S \Rightarrow (R \bowtie_A S) \bowtie_A S$  or  $R \bowtie_A (S \bowtie_A R)$  or  $(R \bowtie_A S) \bowtie_A (S \bowtie_A R)$

CS 347 Notes 03 64

Example:  $R \bowtie S$

	A	B		A	C
R	2	a	S	3	x
	10	b		10	y
	25	c		15	z
	30	d		25	w
				32	x

CS 347 Notes 03 65

Example:  $R \bowtie S$

	A	B		A	C
R	2	a	S	3	x
	10	b		10	y
	25	c		15	z
	30	d		25	w
				32	x

Ans:  $R \bowtie S$

$\Pi_A R = [2, 10, 25, 30]$

$S \bowtie R =$

A	C
10	y
25	w

CS 347 Notes 03 66

	A	B
R	2	a
	10	b
	25	c
	30	d

	A	C
S	3	x
	10	y
	15	z
	25	w
	32	x

Computing transmitted data in example

- with semi-join  $R \bowtie (S \bowtie R)$ :  
 $T = 4 |A| + 2 |A + C| + \text{result}$
- with join  $R \bowtie S$ :  
 $T = 4 |A + B| + \text{result}$

better if  
|B| is large

CS 347                      Notes 03                      67

☛ In general:

- Say R is smaller relation
- $(R \bowtie_A S) \bowtie_A S$  better than  $R \bowtie_A S$  if  
 $\text{size}(\Pi_A S) + \text{size}(R \bowtie_A S) < \text{size}(R)$

CS 347                      Notes 03                      68

- Similar comparisons for other semi-joins
- Remember: only taking into account transmission cost

CS 347                      Notes 03                      69

- Trick:  
 Encode  $\Pi_A S$  (or  $\Pi_A R$ ) as a bit vector

key in S →

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

← one bit/possible key →

CS 347                      Notes 03                      70

Three way joins with semi-joins

Goal:  $R \bowtie S \bowtie T$

CS 347                      Notes 03                      71

Three way joins with semi-joins

Goal:  $R \bowtie S \bowtie T$

Option 1:  $R' \bowtie S' \bowtie T$   
 where  $R' = R \bowtie S$ ;  $S' = S \bowtie T$

Option 2:  $R'' \bowtie S' \bowtie T$   
 where  $R'' = R \bowtie S'$ ;  $S' = S \bowtie T$

CS 347                      Notes 03                      72

- Many options
- Number of semi-join options is exponential in # of relations in join

CS 347

Notes 03

73

### Privacy-preserving join

- Site 1 has R(A,B)
- Site 2 has S(A,C)
- Want to compute  $R \bowtie S$
- Site 1 should NOT discover any S info not in the join
- Site 2 should NOT discover any R info not in the join



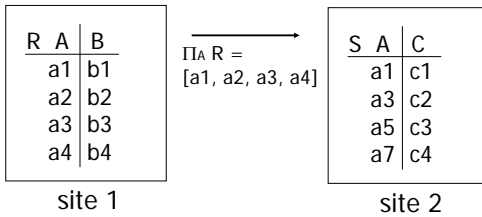
CS 347

Notes 03

74

### Semi-join does not work

- If site 1 sends  $\Pi_A R$  to site 2, site 2 learns all keys of R



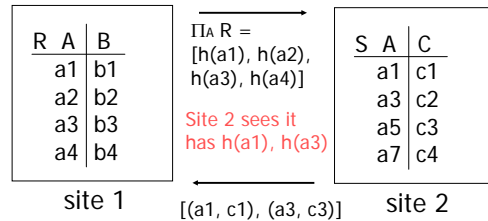
CS 347

Notes 03

75

### Fix: send hashed keys

- Site 1 hashes each value of A before sending
- Site 2 hashes (same function) its own A values to see what tuples match

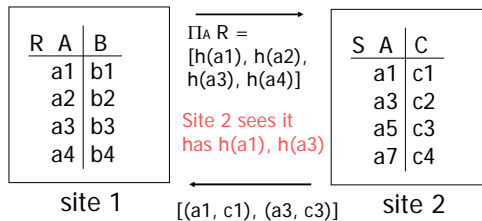


CS 347

Notes 03

76

### What is problem?



- Dictionary attack  
Site 2 takes all keys a1, a2, a3, ... and checks if h(a1), h(a2), h(a3), ... matches what site 1 sent

CS 347

Notes 03

77

### Adversary model

- Honest but curious
  - dictionary attack is possible
  - sending incorrect keys not possible

CS 347

Notes 03

78

### One solution [Agrawal et al.]

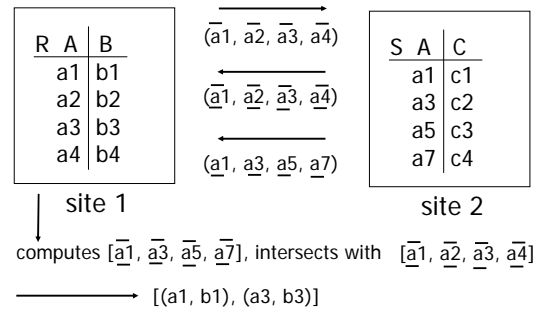
- Use commutative encryption function
  - $E_1(x) = x$  encryption using site i's private key
  - $E_1(E_2(x)) = E_2(E_1(x))$
  - Shorthand for example:
    - $E_1(x)$  is  $\bar{x}$
    - $E_2(x)$  is  $\underline{x}$
    - $E_1(E_2(x))$  is  $\bar{\underline{x}}$

CS 347

Notes 03

79

### Solution



CS 347

Notes 03

80

- Why does this solution work?

CS 347

Notes 03

81

### Other parallel operations

- Duplicate elimination
  - Sort first (in parallel)
  - then eliminate duplicates in result
  - Partition tuples (range or hash) and eliminate locally
- Aggregates
  - Partition by grouping attributes; compute aggregate locally

CS 347

Notes 03

82

### Example

Ra

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

Rb

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

- sum(sal) group by dept

CS 347

Notes 03

83

### Example

Ra

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

Rb

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

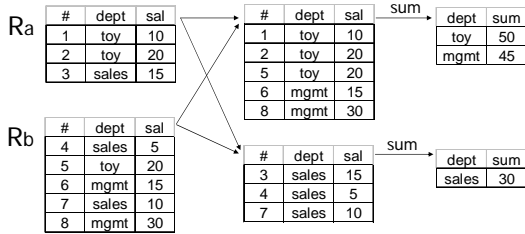
- sum(sal) group by dept

CS 347

Notes 03

84

### Example



- sum(sal) group by dept

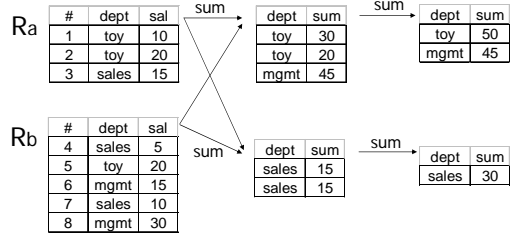
CS 347

Notes 03

85

### Example

less data!



- sum(sal) group by dept

CS 347

Notes 03

86

### Enhancements for aggregates

- Perform aggregate during partition to reduce data transmitted
- Does not work for all aggregate functions...

Which ones?

CS 347

Notes 03

87

### Selection

- Range or hash partition
- Straightforward

But what about indexes?

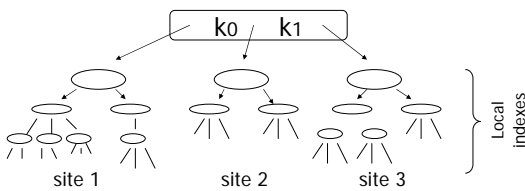
CS 347

Notes 03

88

### Indexing

- Can think of partition vector as root of distributed index:

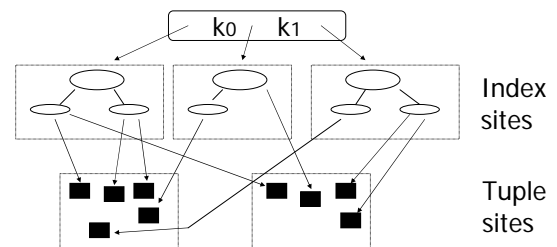


CS 347

Notes 03

89

### Index on non-partition attribute



CS 347

Notes 03

90

## Notes

- If index is not too big, it may be better to keep whole and make copies
- If updates are frequent, can partition update work

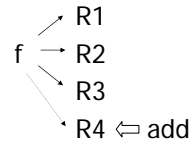
How do we handle the split of B-tree pages?

CS 347

Notes 03

91

- Extensible or linear hashing



CS 347

Notes 03

92

- How do we adapt schemes?
- Where do we store directory, set of participants, ...?
- Which one is better for a distributed environment?
- Can we design a hashing scheme with no global knowledge (P2P)?

CS 347

Notes 03

93

## Summary: query processing

- Decomposition and localization ✓
- Optimization
  - Overview ✓
  - Tricks for joins, sort, ... ✓
  - Tricks for inter-operations parallelism
  - Strategies for optimization

CS 347

Notes 03

94

## Inter-operation parallelism

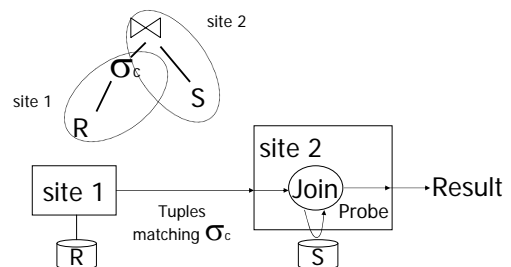
- Pipelined
- Independent

CS 347

Notes 03

95

## Pipelined parallelism

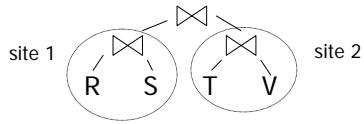


CS 347

Notes 03

96

## Independent parallelism



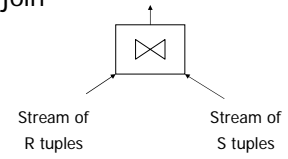
- (1)  $\text{temp1} \leftarrow R \bowtie S$ ;  $\text{temp2} \leftarrow T \bowtie V$
- (2)  $\text{result} \leftarrow \text{temp1} \bowtie \text{temp2}$

CS 347

Notes 03

97

- Pipelining cannot be used in all cases  
E.g., hash join



CS 347

Notes 03

98

## Summary

- As we consider query plans for optimization, we must consider various tricks:
  - for individual operations
  - for scheduling operations

CS 347

Notes 03

99