Big data (RDBMS-style)

Greenplum

aster data

VERTICA

VOLTDB

SMART DATA FAST
Query Processing

- Decomposition
- Localization
- Optimization
Decomposition

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

- Convert from general language to a “standard” form (e.g., Relational Algebra)
Example

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

\[\pi_{A, C} \sigma_{(R.B=1 \lor R.C>3) \land S.D.=2}\]

Conjunctive normal form
Also: Detect invalid expressions

E.g.: Select * from R where R.A =3

⇒ R does not have “A” attribute
Eliminate redundancy

E.g.: in conditions:

\[(S.A=1) \land (S.A>5) \implies \text{False}\]
\[(S.A<10) \land (S.A<5) \implies S.A<5\]
E.g.: Common sub-expressions
Algebraic rewriting

E.g.: Push conditions down
• After decomposition:
  – One or more algebraic query trees on relations

• Localization:
  – Replace relations by corresponding fragments
Localization steps

(1) Start with query  
(2) Replace relations by fragments  
(3) Push $\bigcup$: up (use CS245 rules)

\[ \pi, \sigma : \text{down} \]

(4) Simplify – eliminate unnecessary operations
Notation for fragment

\([R: \text{cond}]\)

- fragment
- conditions its tuples satisfy
Example A

\( (1) \quad \sigma_{E=3}^E \quad | \quad R \)
$(2)$

$$\sigma_{E=3} \cup [R_1: E < 10] \cup [R_2: E \geq 10]$$
(3) \[ \bigcup \sigma_{E=3} \bigcup \sigma_{E=3} \]

\[ [R_1: E < 10] \quad [R_2: E \geq 10] \]
(3)

\[ \sigma_{E=3} \bigcup \sigma_{E=3} \]

\[ [R_1: E < 10] \bigcup [R_2: E \geq 10] \]

\[ \Rightarrow \emptyset \]
\( (4) \quad \sigma_{E=3} \quad \mid \quad [R_1: E < 10] \)
Rule 1

\( \sigma_{c_1}[R: c2] \Rightarrow \sigma_{c_1}[R: c1 \land c2] \)

\( [R: \text{False}] \Rightarrow \emptyset \)
In example A:

\[ \sigma_{E=3}[R_2: E \geq 10] \Rightarrow \sigma_{E=3} [R_2: E=3 \land E \geq 10] \]

\[ \Rightarrow \sigma_{E=3} [R_2: \text{False}] \]

\[ \Rightarrow \emptyset \]
Example B

(1)  

$A$=common attribute

\[ A \quad R \quad S \]
(2) $A \cup \left[ R_1: A < 5 \right] \cup \left[ R_2: 5 \leq A \leq 10 \right] \cup \left[ R_3: A > 10 \right] \cup \left[ S_1: A < 5 \right] \cup \left[ S_2: A \geq 5 \right]$
(3)

\[ \bigcup \]

\[ [R_1: A < 5][S_1: A < 5] \quad [R_1: A < 5][S_2: A \geq 5] \quad [R_2: 5 \leq A \leq 10][S_1: A < 5] \]

\[ [R_2: 5 \leq A \leq 10][S_2: A \geq 5] \quad [R_3: A > 10][S_1: A < 5] \quad [R_3: A > 10][S_2: A \geq 5] \]
(3) \[ (R_1: A < 5) \cup (R_2: 5 \leq A \leq 10) \cup (R_3: A > 10) \]

\[
[R_1: A < 5][S_1: A < 5] \quad [R_1: A < 5][S_2: A \geq 5] \quad [R_2: 5 \leq A \leq 10][S_1: A < 5] \\
[R_2: 5 \leq A \leq 10][S_2: A \geq 5] \quad [R_3: A > 10][S_1: A < 5] \quad [R_3: A > 10][S_2: A \geq 5]
\]
(4) \[ U \]

\[ [R_1: A < 5][S_1: A < 5] \quad [R_2: 5 \leq A \leq 10][S_2: A \geq 5] \]

\[ [R_3: A > 10][S_2: A \geq 5] \]
Rule 2

\[ [R: C_1] \otimes A \quad [S: C_2] \Rightarrow \]

\[ [R \otimes S: C_1 \land C_2 \land R.A = S.A] \]
In step 4 of Example B:

\[ [R_1: A < 5] \Join_A [S_2: A \geq 5] \]

\[ \Rightarrow [R_1 \Join_A S_2: R_1.A < 5 \land S_2.A \geq 5 \land R_1.A = S_2.A ] \]

\[ \Rightarrow [R_1 \Join_A S_2: \text{False}] \Rightarrow \emptyset \]
Localization with derived fragmentation

Example C

(2)

\[
\begin{align*}
U & \cup \quad K \quad U \\
R_1: & A < 10 \\
R_2: & A \geq 10 \\
S_1: & K = R.K \\
& \land R.A < 10 \\
S_2: & K = R.K \\
& \land R.A \geq 10
\end{align*}
\]
\[(3) \quad \bigcup \quad \begin{array}{cccc}
\text{K} & \text{K} & \text{K} & \text{K} \\
\text{[R}_1\text{][S}_1\text{]} & \text{[R}_1\text{][S}_2\text{]} & \text{[R}_2\text{][S}_1\text{]} & \text{[R}_2\text{][S}_2\text{]} \\
\end{array}\]
(4)

\[ \bigcup \left[ R_1: A < 10 \right. \left. \begin{align*} S_1: K &= R.K \\ &\land R.A < 10 \end{align*} \right] \quad \bigcup \left[ R_2: A \geq 10 \right. \left. \begin{align*} S_2: K &= R.K \\ &\land R.A \geq 10 \end{align*} \right] \]
In step 4 of Example C:

\[ [R_1: A < 10] \uplus_K [S_2: K = R.K \land R.A \geq 10] \]
\[ \Rightarrow [R_1 \uplus_K S_2: R_1.A < 10 \land S_2.K = R.K \land R.A \geq 10 \land R_1.K = S_2.K] \]
\[ \Rightarrow [R_1 \uplus_K S_2: \text{False}] \quad (K \text{ is key of } R, \ R_1) \]
\[ \Rightarrow \emptyset \]
(4) simplified more:

\[ \bigcup K \bigcup K \]

\[ [R_1: A < 10] \left( S_1: K = R \cdot K \wedge R \cdot A < 10 \right) \]

\[ [R_2: A \geq 10] \left( S_2: K = R \cdot K \wedge R \cdot A \geq 10 \right) \]
• Localization with vertical fragmentation

Example D

(1) \[\Pi_A \quad \begin{cases} R_1(K, A, B) \\ R_2(K, C, D) \end{cases}\]
(2) \[ \Pi_A \]

\[ \begin{array}{c}
R_1 \\
R_2 \\
(K, A, B) \\
(K, C, D)
\end{array} \]

\[ K \]
(3) \[ \Pi_A \]

\[ \Pi_{K,A} \]

\[ R_1 \quad \text{(K, A, B)} \]

\[ \Pi_{K,A} \]

\[ R_2 \quad \text{(K, C, D)} \]

\[ K \]

not really needed
(4) \[ \Pi_A \]
\[ \downarrow \]
\[ R_1 \]
\((K, A, B)\)
Rule 3

• Given vertical fragmentation of $R$:
  \[ R_i = \Pi_{A_i} (R), \ A_i \subseteq A \]

• Then for any $B \subseteq A$:
  \[ \Pi_B (R) = \Pi_B \left[ \bigcap_{i} R_i \mid B \cap A_i \neq \emptyset \right] \]
• Localization with hybrid fragmentation

Example E

\[ R_1 = \sigma_{k<5} [\Pi_{k,A} \ R] \]

\[ R_2 = \sigma_{k\geq5} [\Pi_{k,A} \ R] \]

\[ R_3 = \Pi_{k,B} \ R \]
Query: \[ \Pi_A \sigma_{k=3} R \]
Reduced Query:

\[ \Pi_A \sigma_{k=3} \]

\[ R_1 \]
Summary - Query Processing

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

Generate query plans
Estimate size of intermediate results
Estimate cost of plan (\$,time,...)

\[ P_1, P_2, P_3, P_n \]
\[ C_1, C_2, C_3, C_n \]

pick minimum
Differences with centralized optimization:

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

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>...</th>
<th>12</th>
<th>...</th>
<th>19</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td>15</td>
<td>20</td>
<td></td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

F₁  F₂  F₃
Basic sort

- $R(K,...)$, sort on $K$
- Fragmented on $K$

Vector: $k_0, k_1, ... k_n$

<table>
<thead>
<tr>
<th></th>
<th>$k_0$</th>
<th></th>
<th>$k_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>10</td>
<td>11</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>17</td>
<td>20</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• **Algorithm**: each fragment sorted independently
• If necessary, ship results
⇒ Same idea on different architectures:

**Shared nothing:**

**Shared memory:**

![Diagram of shared nothing and shared memory architectures]
Range partitioning sort

- $R(K,\ldots)$, sort on $K$
- $R$ located at one or more site/disk, not fragmented on $K$
• Algorithm:
  (a) Range partition on K
  (b) Basic sort

Ra → R1 → R’1
Rb → R2 → R’2
R3 → R’3

Local sort

Result
• Selecting a good partition vector

\[ \begin{array}{c|c|c|c|c}
7 & \ldots & 31 & \ldots & 10 \\
52 & 8 & 12 & 4 \\
11 & 15 & 11 & \\
14 & 32 & 17 & \\
\end{array} \]

\[ Ra \quad Rb \quad Rc \]
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)
• **Sample scenario:**

Coordinator receives:

- S\textsubscript{A}: Min=5  Max=10  \# = 10 tuples
- S\textsubscript{B}: Min=7  Max=17  \# = 10 tuples
• **Sample scenario:**

Coordinator receives:

- $S_A$: Min=5 Max=10 # = 10 tuples
- $S_B$: Min=7 Max=17 # = 10 tuples

**Expected tuples:**

- 5
- 10
- 15
- 20

- \(\text{ko?}\) [assuming we want to sort at 2 sites]
Expected tuples:

5  10  15  20

ko?  [assuming we want to sort at 2 sites]
Expected tuples: = Total tuples
with key < ko = 2

2(ko - 5) + (ko - 7) = 10
3ko = 10 + 10 + 7 = 27
ko = 9

[assuming we want to sort at 2 sites]
Variations

- Send more info to coordinator
  - Partition vector for local site
    Eg. Sa: \[\begin{array}{cccc}
      3 & 3 & 3 & \text{# tuples} \\
      5 & 6 & 8 & 10 & \text{local vector}
    \end{array}\]

- Histogram
More than one round

E.g.: (1) Sites send range and # tuples
      (2) Coordinator returns “preliminary” vector Vo
      (3) Sites tell coordinator how many tuples in each Vo range
      (4) Coordinator computes final vector Vf
Can you come up with a distributed algorithm?

(no coordinator)
Parallel external sort-merge

- Same as range-partition sort, except sort first

Ra → Local sort → Ra' → R1
Rb → Local sort → Rb' → R2

Result

In order

Merge

R3 → R2 → R1 → Result
Parallel external sort-merge

- Same as range-partition sort, except sort first

Ra  \[\text{Local sort}\]  Ra'  \[\rightarrow\]  R1
Rb  \[\text{Local sort}\]  Rb'  \[\rightarrow\]  R2

\[\text{In order}\]

\[\text{Merge}\]

Result

Note: can use merging network if available (e.g., Teradata)
• **Parallel/distributed Join**

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

**Output:** R $\bowtie$ S  
Result at one or more sites
Partitioned Join (Equi-join)
Notes:

- Same partition function $f$ is used for both $R$ and $S$ (applied to join attribute)
- $f$ can be range or hash partitioning
- Local join can be of any type (use any CS245 optimization)
- Various scheduling options e.g.,
  - (a) partition $R$; partition $S$; join
  - (b) partition $R$; build local hash table for $R$; partition $S$ and join
More notes:

• We already know why part-join works:

  R1  R2  R3  S1  S2  S3
  U    U    U

  ⇒

  U
  R1  S1  R2  S2  R3  S3

• Useful to give this type of join a name, because we may want to partition data to make partition-join possible

  (especially in parallel DB system)
Even more notes:

- Selecting good partition function $f$ very important:
  - Number of fragments
  - Hash function
  - Partition vector
• Good partition vector
  – Goal: $|R_i| + |S_i|$ the same
  – Can use coordinator to select
Asymmetric fragment + replicate join

\[ Ra \rightarrow R_1 \rightarrow S \rightarrow Sa \]
\[ Rb \rightarrow R_2 \rightarrow S \rightarrow Sb \]
\[ R_3 \rightarrow S \rightarrow \text{Result} \]
\[ f \rightarrow \text{partition} \]
\[ \text{Result} \rightarrow \text{union} \]
Notes:

• Can use any partition function \( f \) for \( R \) (even round robin)
• Can do any join — not just equi-join

  e.g.: \( R \bowtie S \)

  \( R.A < S.B \)
General fragment and replicate join

partition

\[ f_1 \]

\[ \rightarrow \] 3 fragments

n copies of each fragment

\text{Ra} \rightarrow \text{R1} \rightarrow \text{R1}

\text{Rb} \rightarrow \text{R2} \rightarrow \text{R2}

\text{R3} \rightarrow \text{R3}
S is partitioned in similar fashion

All $n \times m$ pairings of $R, S$ fragments

Result
Notes:

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

  $(R \bowtie_A S) \bowtie_A (S \bowtie_A R)$
Example: $R \bowtie S$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>a</td>
</tr>
<tr>
<td>10</td>
<td>b</td>
</tr>
<tr>
<td>25</td>
<td>c</td>
</tr>
<tr>
<td>30</td>
<td>d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>x</td>
</tr>
<tr>
<td>10</td>
<td>y</td>
</tr>
<tr>
<td>15</td>
<td>z</td>
</tr>
<tr>
<td>25</td>
<td>w</td>
</tr>
<tr>
<td>32</td>
<td>x</td>
</tr>
</tbody>
</table>
Example: $R \bowtie S$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>a</td>
</tr>
<tr>
<td>10</td>
<td>b</td>
</tr>
<tr>
<td>25</td>
<td>c</td>
</tr>
<tr>
<td>30</td>
<td>d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>x</td>
</tr>
<tr>
<td>10</td>
<td>y</td>
</tr>
<tr>
<td>15</td>
<td>z</td>
</tr>
<tr>
<td>25</td>
<td>w</td>
</tr>
<tr>
<td>32</td>
<td>x</td>
</tr>
</tbody>
</table>

$\Pi_A R = [2, 10, 25, 30]$
Example: $R \bowtie S$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>2</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>d</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>3</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>y</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>z</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>w</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>x</td>
</tr>
</tbody>
</table>

$R \bowtie S =$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>y</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>w</td>
</tr>
</tbody>
</table>
Computing transmitted data in example:

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

• with join \( R \bowtie S \):
  \[ T = 4 \ |A+B| + \text{result} \]
Computing transmitted data in example:

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

- with join $R \bowtie S$:
  \[ T = 4 |A+B| + \text{result} \]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>2</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>3</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>y</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>z</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>w</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>x</td>
</tr>
</tbody>
</table>

Better if say $|B|$ is large
In general:

- Say $R$ is smaller relation
- $(R \times_A S) \Join_A S$ better than $R \Join_A S$ if

$$\text{size } (\Pi_A S) + \text{size } (R \times_A S) < \text{size } (R)$$
• Similar comparisons for other semi-joins
• \textbf{Remember:} only taking into account transmission cost
• 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------->
Three way joins with semi-joins

Goal: R \( \bowtie \) S \( \bowtie \) T
Three way joins with semi-joins

Goal: $R \bowtie S \bowtie T$

Option 1: $R' \bowtie S' \bowtie T$

where $R' = R \bowtie S$; $S' = S \bowtie T$
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 \( R'' = R \bowtie S' \); \( S' = S \bowtie T \)
Many options!

Number of semi-join options is exponential in # of relations in join
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
Semi-Join Does Not Work

• If Site 1 sends $\Pi_A R$ to Site 2, site 2 leans all keys of $R$!

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td></td>
</tr>
<tr>
<td>a3</td>
<td>b3</td>
<td></td>
</tr>
<tr>
<td>a4</td>
<td>b4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>c1</td>
<td></td>
</tr>
<tr>
<td>a3</td>
<td>c2</td>
<td></td>
</tr>
<tr>
<td>a5</td>
<td>c3</td>
<td></td>
</tr>
<tr>
<td>a7</td>
<td>c4</td>
<td></td>
</tr>
</tbody>
</table>

site 1

$\Pi_A R = (a1, a2, a3, a4)$

site 2
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

\[ \Pi_A R = (h(a_1), h(a_2), h(a_3), h(a_4)) \]

Site 2 sees it has \( h(a_1), h(a_3) \)

\[ (a_1, c_1), (a_3, c_3) \]
What is problem?

\[
\Pi_A R = (h(a_1), h(a_2), h(a_3), h(a_4))
\]

Site 2 sees it has \( h(a_1), h(a_3) \)

(a_1, c_1), (a_3, c_3)
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...

\[
\Pi_A R = (h(a1), h(a2), h(a3), h(a4))
\]

Site 2 sees it has $h(a1), h(a3)$

\[
(a1, c1), (a3, c3)
\]
Adversary Model

• Honest but Curious
  – dictionary attack is possible (cheating is internal and can’t be caught)
  – sending incorrect keys not possible (cheater could be caught)
One Solution (Agrawal et al)

- Use commutative encryption function
  - $E_i(x) = x$ encryption using site $i$ private key
  - $E_1(E_2(x)) = E_2(E_1(X))$
  - Shorthand for example:
    - $E_1(x)$ is $\overline{x}$
    - $E_2(x)$ is $x$
    - $E_1(E_2(x))$ is $\overline{x}$
Solution:

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td></td>
</tr>
<tr>
<td>a3</td>
<td>b3</td>
<td></td>
</tr>
<tr>
<td>a4</td>
<td>b4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>c1</td>
<td></td>
</tr>
<tr>
<td>a3</td>
<td>c2</td>
<td></td>
</tr>
<tr>
<td>a5</td>
<td>c3</td>
<td></td>
</tr>
<tr>
<td>a7</td>
<td>c4</td>
<td></td>
</tr>
</tbody>
</table>

site 1 computes \((\overline{a_1}, \overline{a_3}, \overline{a_5}, \overline{a_7})\), intersects with \((\overline{a_1}, \overline{a_2}, \overline{a_3}, \overline{a_4})\)

\((a_1, b_1), (a_3, b_3)\)
Why does this solution work?
Other Privacy Preserving Operations?

• Inequality join \( R \bowtie S \quad R.A > S.A \)

• Similarity Join \( R \bowtie S \quad \text{sim}(R.A,S.A) < e \)
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
Example:

<table>
<thead>
<tr>
<th>#</th>
<th>dept</th>
<th>sal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>toy</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>sales</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>dept</th>
<th>sal</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>sales</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>toy</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>mgmt</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>sales</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>mgmt</td>
<td>30</td>
</tr>
</tbody>
</table>

- sum (sal) group by dept
Example:

\[ \text{sum} \ (\text{sal}) \ \text{group by dept} \]
Example:

\[
\begin{array}{|c|c|c|}
\hline
\# & \text{dept} & \text{sal} \\
1 & toy & 10 \\
2 & toy & 20 \\
3 & sales & 15 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\# & \text{dept} & \text{sal} \\
4 & sales & 5 \\
5 & toy & 20 \\
6 & mgmt & 15 \\
7 & sales & 10 \\
8 & mgmt & 30 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\# & \text{dept} & \text{sal} \\
1 & toy & 10 \\
2 & toy & 20 \\
5 & toy & 20 \\
6 & mgmt & 15 \\
8 & mgmt & 30 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\# & \text{dept} & \text{sal} \\
3 & sales & 15 \\
4 & sales & 5 \\
7 & sales & 10 \\
\hline
\end{array}
\]

- \text{sum (sal) group by dept}

\[
\begin{array}{|c|c|}
\hline
\text{dept} & \text{sum} \\
toy & 50 \\
mgmt & 45 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{dept} & \text{sum} \\
sales & 30 \\
\hline
\end{array}
\]
Example:

<table>
<thead>
<tr>
<th>#</th>
<th>dept</th>
<th>sal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>toy</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>sales</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>dept</th>
<th>sal</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>sales</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>toy</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>mgmt</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>sales</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>mgmt</td>
<td>30</td>
</tr>
</tbody>
</table>

• sum (sal) group by dept
Example:

<table>
<thead>
<tr>
<th>#</th>
<th>dept</th>
<th>sal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>toy</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>sales</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>dept</th>
<th>sal</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>sales</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>toy</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>mgmt</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>sales</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>mgmt</td>
<td>30</td>
</tr>
</tbody>
</table>

- sum (sal) group by dept
Example:

\[
\begin{array}{|c|c|c|}
\hline
# & dept & sal \\
\hline
1 & toy & 10 \\
2 & toy & 20 \\
3 & sales & 15 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
# & dept & sal \\
\hline
4 & sales & 5 \\
5 & toy & 20 \\
6 & mgmt & 15 \\
7 & sales & 10 \\
8 & mgmt & 30 \\
\hline
\end{array}
\]

- \text{sum (sal) group by dept}

less data!
Preview: Map Reduce

data A1

data A2

data A3

data B1

data B2

data C1

data C2
Enhancements for aggregates

• Perform aggregate during partition to reduce data transmitted
• Does not work for all aggregate functions...
  Which ones?
**Selection**

- Range or hash partition
- Straightforward

But what about **indexes**?
Indexing

- Can think of partition vector as root of distributed index:
• Index on non-partition attribute
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...
  (Question: how do we handle split of B-Tree pages?)
• Extensible or linear hashing

```
R1
f → R2
    ↓
R3
    ↓
R4  <- add
```
• 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)?
Summary: Query processing

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

- Pipelined
- Independent
Pipelined parallelism

Site 1

Site 2

Site 1

R

S

σ_c

Join

Probe

result

Tuples matching σ_c
Independent parallelism

Site 1
R \rightarrow S

Site 2
T \rightarrow V

(1) \text{temp1} \leftarrow R \circlearrowleft S; \quad \text{temp2} \leftarrow T \circlearrowleft V

(2) \text{result} \leftarrow \text{temp1} \circlearrowright \text{temp2}
- Pipelining cannot be used in all cases e.g.: Hash Join
Summary

As we consider query plans for optimization, we must consider various tricks:

- for individual operations
- for scheduling operations