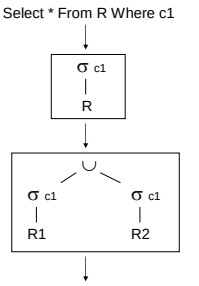
Query Processing

Query Processing

- Decomposition
- Localization
- Optimization
  - Global
  - Local



Optimal processing strategy
•Ship R1, R2 to result site, then select
•Perform select at R1 site, R2 site, then ship result

Decomposition

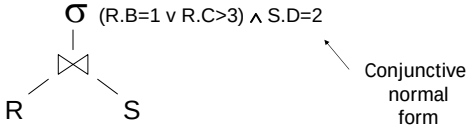
- Same as in centralized system
  - Normalization
  - Eliminating redundancy
  - Algebraic rewriting

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)

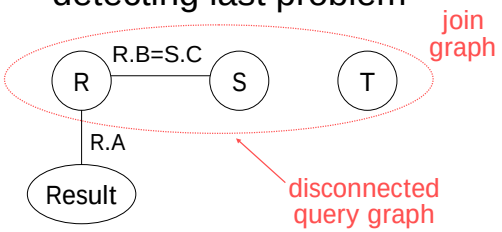


Also: Detect invalid expressions

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

Note: Query graph useful for detecting last problem



Eliminate redundancy

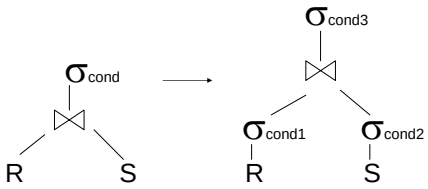
E.g.: in conditions:

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

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

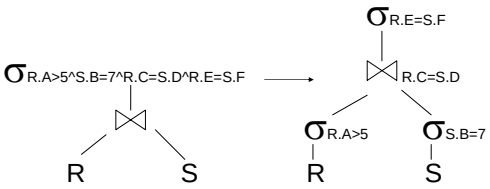
Algebraic rewriting

E.g.: Push conditions down



Algebraic rewriting

E.g.: Push conditions down



- After decomposition:
  - algebraic query tree on relations
- Localization:
  - Replace relations by corresponding fragments

Localization steps

(1) Start with query

(2) Replace relations by fragments

(3) Push $\cup$: up (use CS3223 rules)

π, σ : down

(4) Simplify – eliminate unnecessary operations

Notation for fragment

[R: cond]

fragment — conditions its tuples satisfy

Example A

(1) $\sigma_{E=3}$

|

R

(2) $\sigma_{E=3}$

|

$\cup$

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

(3)

$\cup$

$\sigma_{E=3}$ $\sigma_{E=3}$

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

$\Rightarrow$ $\emptyset$

(4) $\sigma_{E=3}$

|

$[R_1: E < 10]$

Rule 1

(A) $\sigma_{C1}[R: c2] \Rightarrow [\sigma_{C1} R: c1 \wedge c2]$

(B) $[R: False] \Rightarrow \emptyset$

In example A:

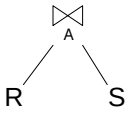
$\sigma_{E=3}[R_2: E \geq 10] \Rightarrow [\sigma_{E=3} R_2: E=3 \wedge E \geq 10]$

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

$\Rightarrow \emptyset$

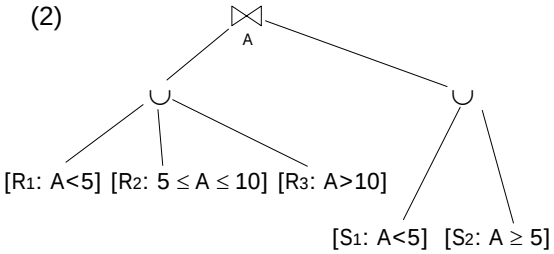
Example B

(1)

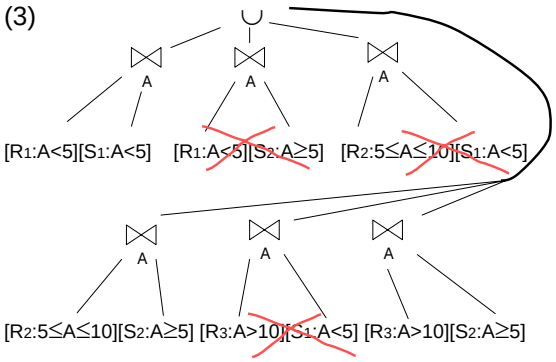


A=common attribute

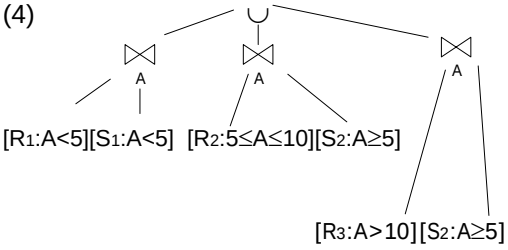
(2)



(3)



(4)



Rule 2

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

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

☛ In step 4 of Example B:

$[R_1: A<5] \underset{A}{\bowtie} [S_2: A \geq 5]$

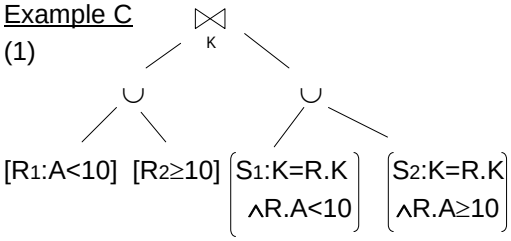
$\Rightarrow [R_1 \underset{A}{\bowtie} S_2: R_1.A < 5 \wedge S_2.A \geq 5 \wedge R_1.A = S_2.A]$

$\Rightarrow [R_1 \underset{A}{\bowtie} S_2: \text{False}] \Rightarrow \emptyset$

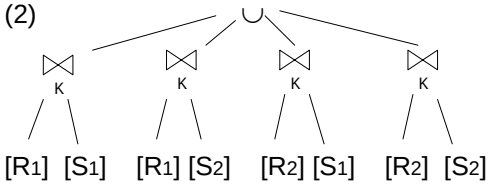
Localization with derived fragmentation

Example C

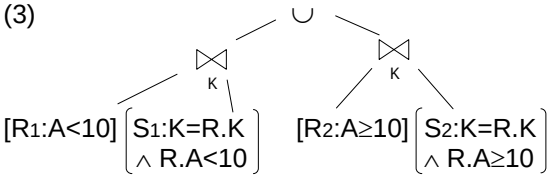
(1)



(2)



(3)



☛ In step 3 of Example C:

$[R_1\text{:}A<10] \underset{K}{\bowtie} [S_2\text{:}K=R.K \wedge R.A\geq 10]$

$\Rightarrow [R_1 \underset{K}{\bowtie} S_2\text{:}\ R_1.A<10 \wedge S_2.K=R.K \wedge R.A\geq 10 \wedge R_1.K=S_2.K]$

$\Rightarrow [R_1 \underset{K}{\bowtie} S_2\text{:False}]$ (K is key of R, R_1)

$\Rightarrow \emptyset$

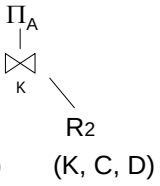
• Localization with vertical fragmentation

Example D

(1) Π_A | R

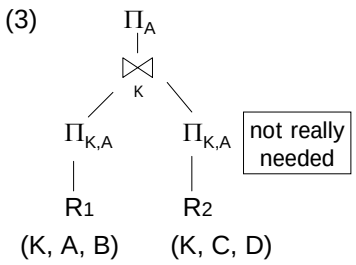
$R_1(K, A, B)$
$R_2(K, C, D)$

(2)

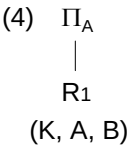


(K, A, B) (K, C, D)

(3)



(4)



Rule 3

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

•Localization with hybrid fragmentation

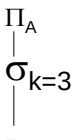
Example E

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

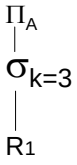
$R_2 = \sigma_{k\geq 5} [\Pi_{k,A} R]$

$R_3 = \Pi_{k,B} R$

Query:



Reduced Query:



Summary - Query Processing

- Decomposition √
- Localization √
- Optimization
  - Tricks for joins
  - Strategies for optimization

Semi-join

- Goal: reduce communication traffic
- $R \underset{A}{\bowtie} S \Rightarrow (R \underset{A}{\ltimes} S) \underset{A}{\bowtie} S$ or

  $R \underset{A}{\bowtie} (S \underset{A}{\ltimes} R)$ or

  $(R \underset{A}{\ltimes} S) \underset{A}{\bowtie} (S \underset{A}{\ltimes} R)$

$$R \underset{A}{\ltimes} S = \Pi_R (R \underset{A}{\bowtie} S)$$

R

A	B
2	a
10	b
25	c
30	d

S

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

Computing transmitted data in example:
- with join $R \bowtie S$:
- Ship R to S (Why?); Evaluate join at S Site; Ship result to query site (assume 3rd site)
- T = 4 |A+B| + result

Example: $R \bowtie S$

R

A	B
2	a
10	b
25	c
30	d

S

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

Ans: $R \bowtie S$

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

$S \ltimes R$ =

A	C
10	y
25	w

R

A	B
2	a
10	b
25	c
30	d

S

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

Computing transmitted data in example:
- with semi-join $R \bowtie (S \ltimes R)$:

T = 4 |A| +2 |A+C| + result

- with join $R \bowtie S$:

T = 4 |A+B| + result

better if say |B| is large

In general:

- Say R is smaller relation
- $(R \underset{A}{\ltimes} S) \underset{A}{\bowtie} S$ better than $R \underset{A}{\bowtie} S$ if

  size (Π_A S) + size ($R \underset{A}{\ltimes} S$) < size (R)

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

Three-way joins with semi-joins

Goal: R ⋈ S ⋈ T

Option 1: R' ⋈ S' ⋈ T
where R' = R ⋉ S; S' = S ⋉ T

Option 2: R'' ⋈ S' ⋈ T
where R'' = R ⋉ S'; S' = S ⋉ T

⇔ Many options!

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

2-way Semijoin

2-way Semijoin – an extended version of the semijoin

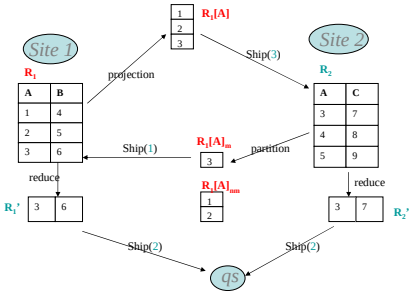
- *Definition:* A 2-way semijoin (t) of R_i and R_j on attribute A can be denoted as
  $R_i \leftarrow A \rightarrow R_j = \{R_i.A \rightarrow R_j,\ R_j.A \rightarrow R_i\}$

So t reduces R_i and R_j to R_i' and R_j' respectively.

2-way Semijoin

- *Computing steps:*
  1) Send $R_i[A]$ from site i to site j ;
  2) Reduce R_j to R_j' by eliminating tuples whose attribute A are not matching any of $R_i[A]$ and at the same time partition $R_j[A]$ to $R_j[A]_m$ (match one of $R_j[A]$) and $R_j[A]_{nm}$($R_j[A]$- $R_j[A]_m$) ;
  3) Send $min(R_i[A]_m, R_j[A]_{nm})$ back to site i ;
  4) Reduce R_i to R_i' using $R_i[A]_m$ (or $R_i[A]_{nm}$) .
  5) Send to R_i' and R_j' join site (3rd site)
- *Evaluation:*
  - S(R) denote size of R
  - Benefit: $B(t) = [S(R_i) - S(R_i')] + [S(R_j) - S(R_j')]$
  - Cost: $C(t) = S(R_i[A]) + min[S(R_i[A]_m), S(R_i[A]_{nm})]$
  - If the benefit exceeds the cost, then it is called a *cost-effective* 2-way semioin

2-way Semijoin



Trick: Encode Π_A R as a bit vector

R

A	B
2	a
10	b
25	c
30	d

S

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

Ans: R⋈S

hash(Π_A R) = [00100000001...]

S ⋉ R =

A	C
10	y
25	w

Pros/Cons?