CSE 562
Database Systems

Query Processing: Algebraic Optimization

Some slides are based or modified from originals by
Database Systems: The Complete Book, Pearson Prentice Hall 2nd Edition

cse@buffalo

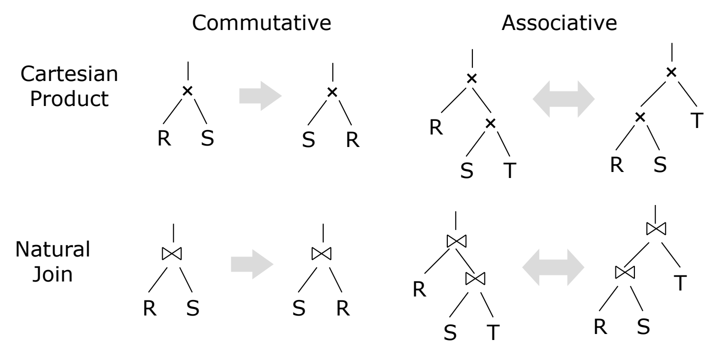
Outline – Query Optimization

- Overview
- Relational algebra level
  - Algebraic Transformations
- Detailed query plan level
  - Estimate Costs
    - Estimating size of results
    - Estimating # of IOs
  - Generate and compare plans

Relational Algebra Optimization

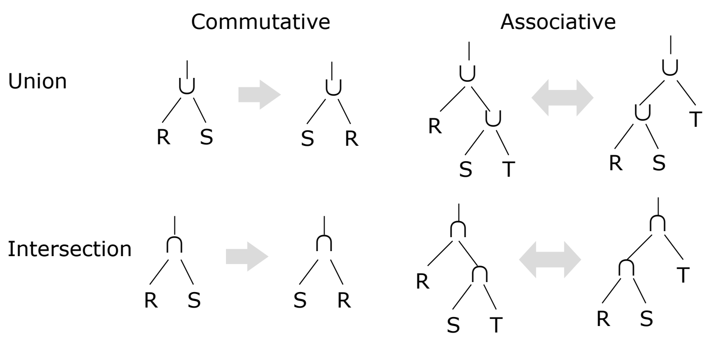
- Transformation rules
  (preserve equivalence)
- What are good transformations?

Algebraic Rewritings: Commutative and Associative Laws



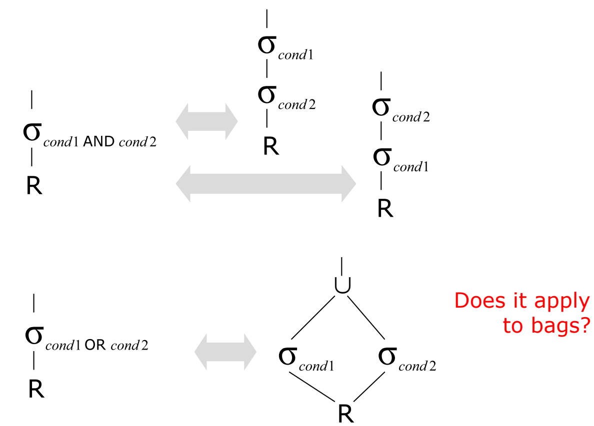
- **Question 1**: Do the above hold for both sets and bags?
- **Question 2**: Do commutative and associative laws hold for arbitrary Theta Joins?

Algebraic Rewritings: Commutative and Associative Laws

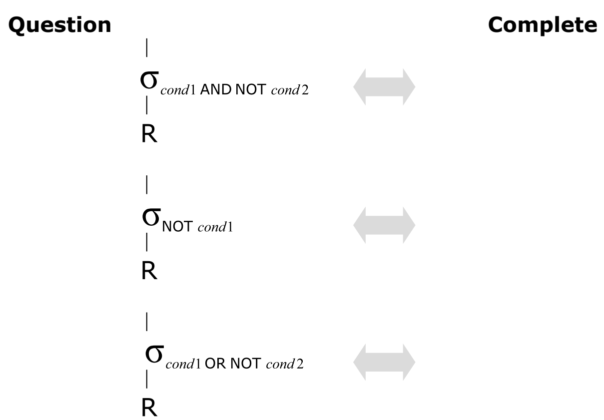


- **Question 1:** Do the above hold for both sets and bags?
- **Question 2:** Is difference commutative and associative?

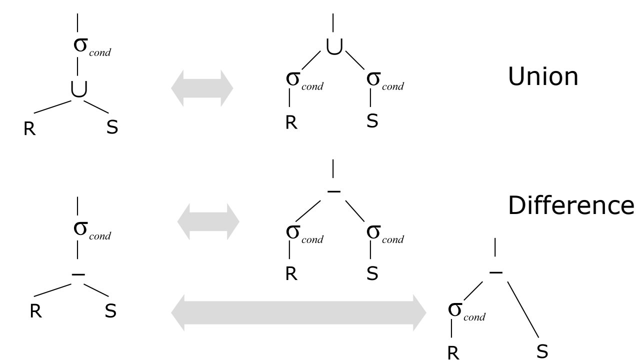
Algebraic Rewritings for Selection: Decomposition of Logical Connectives



Algebraic Rewritings for Selection: Decomposition of Negation

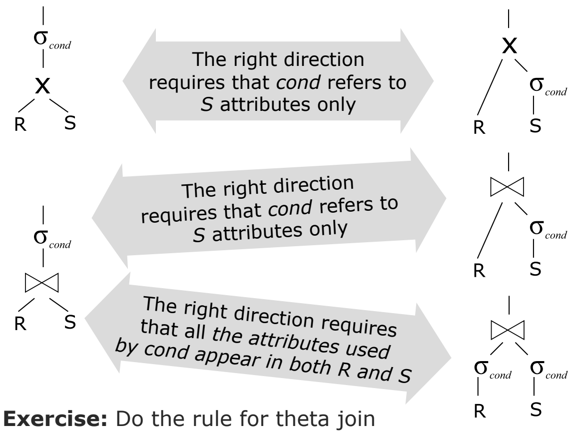


Pushing Selection Through Binary Operators: Union and Difference



- **Exercise:** Do the rules for intersection

Pushing Selection Through Cartesian Product and Join



- **Exercise:** Do the rule for theta join

Rules: π + σ combined

Let **X** = subset of R attributes
Z = attributes in predicate P (subset of R attributes)

$$\pi_X[\sigma_P(R)] = \boxed{\pi_X}\{\sigma_P[\overset{\boxed{\pi_{XZ}}}{\cancel{\pi_X}}(R)]\}$$

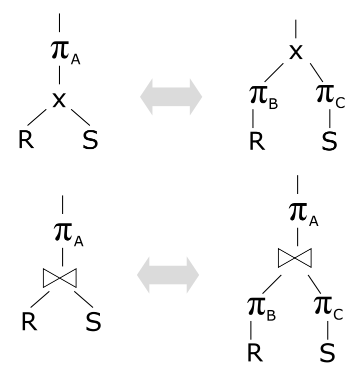
Pushing Simple Projections Through Binary Operators: Union

- A projection is simple if it only consists of an attribute list

Union

- **Question 1:** Does the above hold for both bags and sets?
- **Question 2:** Can projection be pushed below intersection and difference?
- Answer for both bags and sets

Pushing Simple Projections Through Binary Operators: Join and Product

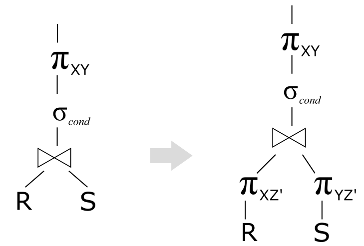


- B is the list of R attributes that appear in A
- Similar for C

- **Question:** What is B and C ?

- **Exercise:** Write the rewriting rule that pushes projection below theta join

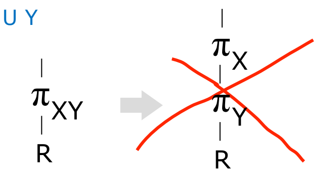
Rules: π + σ + $\bowtie$ combined



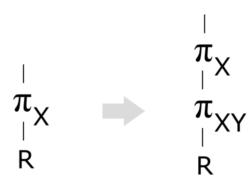
- Z′ = Z U {attributes used in *cond*}

Projection Decomposition

- Let X = set of attributes
- Y = set of attributes
- XY = X U Y



Projection Decomposition



Some Rewriting Rules Related to Aggregation: SUM

- $\sigma_{cond}[SUM_{GroupbyList;GroupedAttribute \to ResultAttribute}(R)]$
  $\Leftrightarrow$
  $SUM_{GroupbyList;GroupedAttribute \to ResultAttribute}[\sigma_{cond}(R)]$
  if *cond* involves only the *GroupbyList*

- $SUM_{GL;GA \to RA}(R \cup S)$
  $\Leftrightarrow$
  $PLUS_{RA1,RA2:RA}[(SUM_{GL;GA \to RA1}R) \bowtie (SUM_{GL;GA \to RA2}S)]$

- $SUM_{GL2;RA1 \to RA2}[SUM_{GL1;GA \to RA1}(R)]$
  $\Leftrightarrow$
  $SUM_{GL2:GA \to RA2}(R)$

- **Question:** does the above hold for both bags and sets?

Derived Rules: σ + $\bowtie$ combined

More Rules can be Derived:

$$\sigma_{p \wedge q}(R \bowtie S) = [\sigma_p(R)] \bowtie [\sigma_q(S)]$$

$$\sigma_{p \wedge q \wedge m}(R \bowtie S) = \sigma_m[\sigma_p(R) \bowtie \sigma_q(S)]$$

$$\sigma_{p \vee q}(R \bowtie S) = [\sigma_p(R) \bowtie S] \cup [R \bowtie \sigma_q(S)]$$

- **p** only at **R**
- **q** only at **S**
- **m** at both **R** and **S**

Derivation for first one

$$\sigma_{p \wedge q}(R \bowtie S) =$$

$$\sigma_p[\sigma_q(R \bowtie S)] =$$

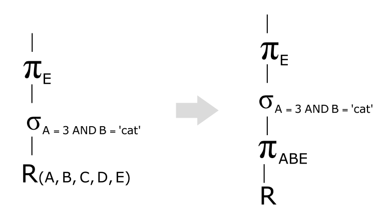
$$\sigma_p[R \bowtie \sigma_q(S)] =$$

$$[\sigma_p(R)] \bowtie [\sigma_q(S)]$$

Which are "good" transformations?

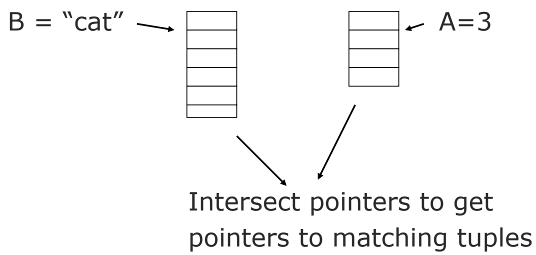
- $\sigma_{p1 \wedge p2}(R) \rightarrow \sigma_{p1}[\sigma_{p2}(R)]$
- $\sigma_p(R \bowtie S) \rightarrow [\sigma_p(R)] \bowtie S$
- $R \bowtie S \rightarrow S \bowtie R$
- $\pi_x[\sigma_p(R)] \rightarrow \pi_x\{\sigma_p[\pi_{xz}(R)]\}$

Conventional Wisdom: Do Projects Early



But…

What if we have A, B indexes?



More Transformations in Textbook

- Eliminate common sub-expressions
- Other operations: duplicate elimination

Bottom line

- No transformation is <u>always</u> good at the logical query plan level
- Usually good:
  - early selections
  - elimination of Cartesian products
  - elimination of redundant sub-expressions
- Many transformations lead to "promising" plans
  - Commuting/rearranging joins
  - In practice too "combinatorially explosive" to be handled as rewriting of logical query plan