Query Optimization

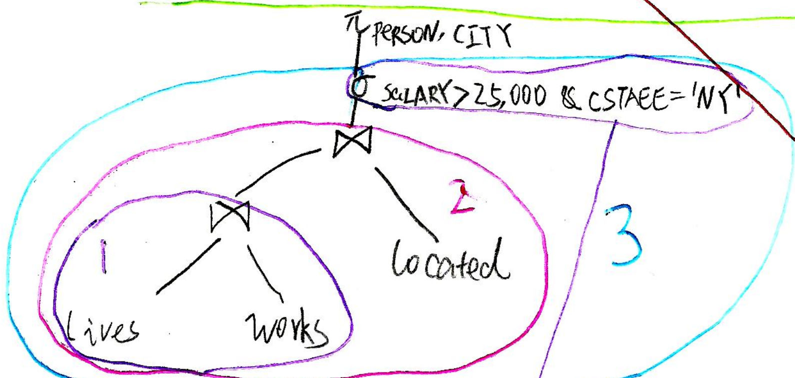
Query Tree

Consider the following schema and query:

lives(PERSON, STREET, CITY, STATE, ZIP)
works(PERSON, COMPANY, SALARY, POSITION)
located(COMPANY, CCITY, CSTATE, CZIP)

Find the name and city of all people who earn more than 25,000 and work for a company located in the state of NY.

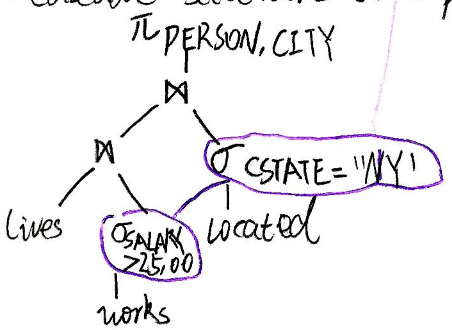
$$\pi_{PERSON, CITY}\ \sigma_{SALARY > 25{,}000 \wedge CSTATE = 'NY'}\ ((lives \bowtie works) \bowtie located)$$



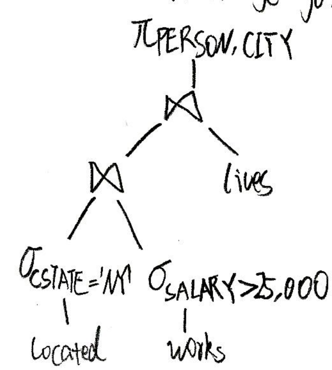
HEURISTIC ALGEBRAIC OPTIMIZATION ALGORITHM

1. Cascade selection (rule 1)
2. Move selection as far down the tree as possible (rules 2, 4, 6, 10)
3. Rearrange leaf node to get smaller intermediate relations (rule 9)
4. Combine a X with σ to yield ⋈, if possible
5. Cascade projections and push down the tree (rules 3, 4, 7, 11)
6. Identify common subexpressions.

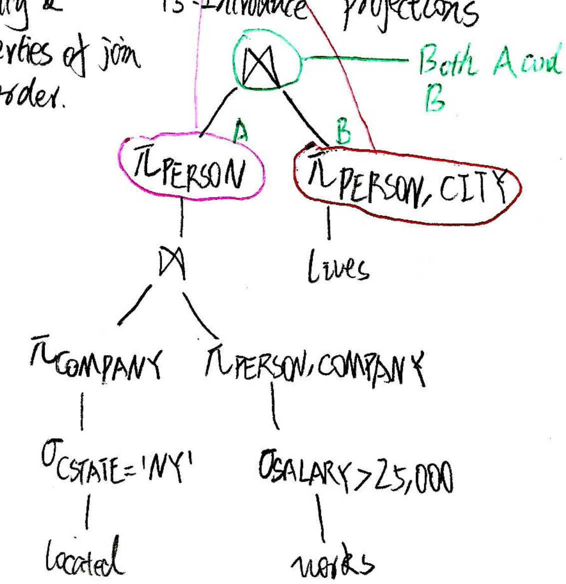
T_1: Cascade selections and push down tree



T_2: Using commutativity & associativity properties of join rearrange join order.



T_3: Introduce projections



CSE 412/598
DATABASE MANAGEMENT
COURSE NOTES

6. QUERY OPTIMIZATION

Department of Computer Science & Engineering
Arizona State University

QUERY TREE

a tree structure representing a relational algebra expression

- leaf node - base relation
- internal node - result of a relational algebra operation

Consider the following schema and query:

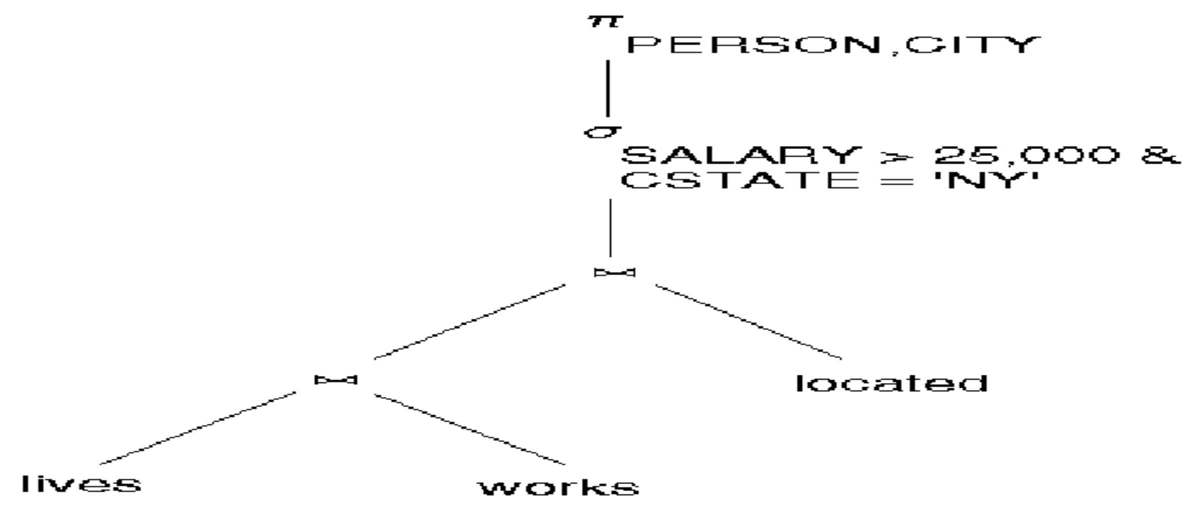
- lives(PERSON,STREET,CITY,STATE,ZIP)
- works(PERSON,COMPANY,SALARY,POSITION)
- located(COMPANY,CCITY,CSTATE,CZIP)

Find the name and city of all people who earn more than 25,000 and work for a company located in the state of NY.

$\pi_{\text{PERSON, CITY}}\ \sigma_{\text{SALARY>25,000} \wedge \text{CSTATE='NY'}}((\text{lives} \bowtie \text{works}) \bowtie \text{located})$

QUERY TREE EXAMPLE

$\pi_{\text{PERSON, CITY}}\ \sigma_{\text{SALARY>25,000} \land \text{CSTATE='NY'}}$ ((lives $\bowtie$ works) $\bowtie$ located)



QUERY OPTIMIZATION

Transform the query as entered by the user into an equivalent query that can be computed more efficiently.

Algebraic expressions may be transformed into equivalent expressions using transformation rules that preserve equivalence. These transformation rules are based on commutative properties of the relational algebra operators and conditions on the expression to be transformed.

TRANSFORMATION RULES

σ and π

1. Cascade of σ:

$$\sigma_{C1 \text{ and } C2 \text{ and } \ldots \text{ and } Cn}(R) \equiv \sigma_{C1}(\sigma_{C2}(\ldots(\sigma_{Cn}(R))\ldots))$$

2. Commutativity of σ:

$$\sigma_{C1}(\sigma_{C2}(R)) \equiv \sigma_{C2}(\sigma_{C1}(R))$$

3. Cascade of π:

$$\pi_{list1}(\pi_{list2}(\ldots(\pi_{listn}(R))\ldots)) = \pi_{list1}(R)$$

4. Commuting σ with π: if C involves only attributes A_1, ..., A_n in the π list, they can be commuted.

$$\pi_{A1, A2, \ldots, An}(\sigma_C(R)) \equiv \sigma_C(\pi_{A1, A2, \ldots, An}(R))$$

TRANSFORMATION RULES
$\bowtie$ and $\times$

5. Commutativity of $\bowtie$ (or $\times$):

$R \bowtie_C S \equiv S \bowtie_C R$

6. Commuting σ with $\bowtie$ (or $\times$):

if the attributes in C involve only the attributes of R:

$$\boldsymbol{\sigma_C (R \bowtie S) \equiv (\sigma_C (R)) \bowtie S}$$

if c is (c1 and c2) and c1 applies to R and c2 to S:

$$\boldsymbol{\sigma_C (R \bowtie S) \equiv (\sigma_{C1} (R)) \bowtie (\sigma_{C2} (S))}$$

TRANSFORMATION RULES

$\bowtie$ and $\times$

7. Commuting π with $\bowtie$ (or $\times$):

Let $A_1, \ldots, A_n$ be attributes of R,

$B_1, \ldots, B_m$ be attributes of S,

$L = \{A_1, \ldots, A_n, B_1, \ldots, B_m\}$

If C involves only the attrs of L:

$$\pi_L (R \bowtie_c S) \equiv (\pi_{A1, \ldots, An} (R)) \bowtie_c (\pi_{B1, \ldots, Bm}(S))$$

If C contains additional attributes not in L, these must be added to the π list and a final π is needed. e.g:

$$\pi_L (R \bowtie_c S) \equiv \pi_L((\pi_{A1,\ldots,An,An+1, \ldots, An+k}(R)) \bowtie_c (\pi_{B1, \ldots, Bm, Bm+1, \ldots, Bm+p} (S)))$$

TRANSFORMATION RULES
$\cup$ and $\cap$

8. Commutativity of set operations: $\cup$ and $\cap$ are commutative.

$(R \cup S) \equiv (S \cup R)$ $(R \cap S) \equiv (S \cap R)$

9. Associativity of $\bowtie$, $\times$, $\cup$, and $\cap$:

They are individually associative.

Let θ be $\bowtie$, $\times$, $\cup$, or $\cap$, then

$$\mathbf{(R\ \theta\ S)\ \theta\ T \equiv R\ \theta\ (S\ \theta\ T)}$$

10. Commuting σ with set operations $\cup$ and $\cap$.

Let θ be $\cup$ or $\cap$, then

$$\sigma_C\ (R\ \theta\ S) \equiv (\sigma_C\ (R))\ \theta\ (\sigma_C\ (S))$$

TRANSFORMATION RULES

$\cup$ and $\cap$

11. Commuting π with set operations:

    The π operation commutes with $\cup$ and $\cap$.

    Let θ be $\cup$ or $\cap$, then

    $\boldsymbol{\pi_L (R\ \theta\ S) \equiv (\pi_L (R))\ \theta\ (\pi_L (S))}$

12. Additional logic transformations:

    $(\forall x)(P(x)) \equiv (\exists x)(\text{not } (P(x)))$

    $(\exists x)(P(x)) \equiv \text{not } (\forall x)(\text{not } (P(x)))$

    $(\forall x)(P(x) \text{ and } Q(x)) \equiv (\exists x)(\text{not } (P(x)) \text{ or not } (Q(x)))$

    $(\forall x)(P(x) \text{ or } Q(x)) \equiv (\exists x)(\text{not } (P(x)) \text{ and not } (Q(x)))$

    $(\exists x)(P(x) \text{ or } Q(x)) \equiv \text{not } (\forall x)(\text{not } (P(x)) \text{ and not } (Q(x)))$

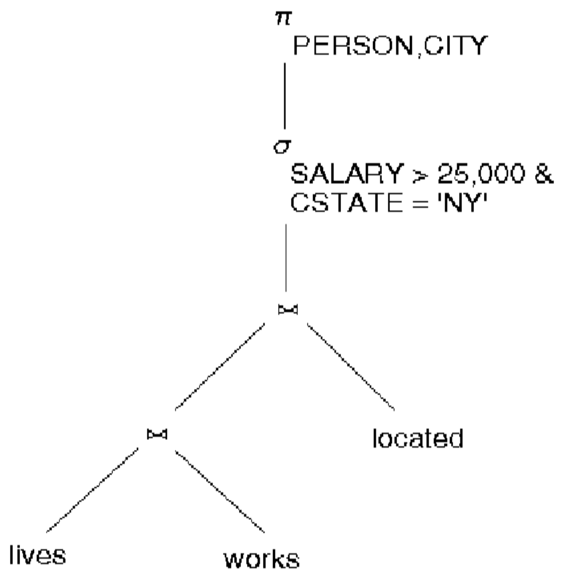
    $(\exists x)(P(x) \text{ and } Q(x)) \equiv \text{not } (\forall x)(\text{not } (P(x)) \text{ or not } (Q(x)))$

HEURISTIC ALGEBRAIC
OPTIMIZATION ALGORITHM

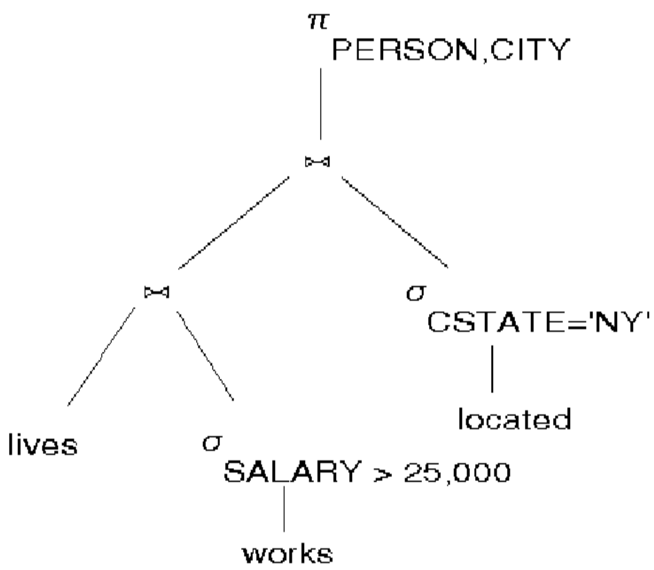
1. Cascade selection
   (rule 1)
2. Move selections as far down the tree as possible
   (rules 2,4,6,10)
3. Rearrange leaf nodes to get smaller intermediate relations (rule 9)
4. Combine a $\times$ with σ to yield a $\bowtie$, if possible
5. Cascade projections and push down the tree
   (rules 3,4,7,11)
6. Identify common subexpressions

QUERY OPTIMIZATION EXAMPLE

T_0: Initial Query Tree

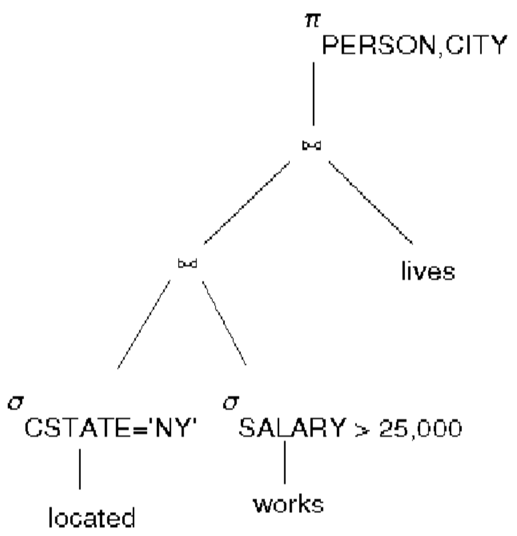


T_1: Cascade selections and push down tree

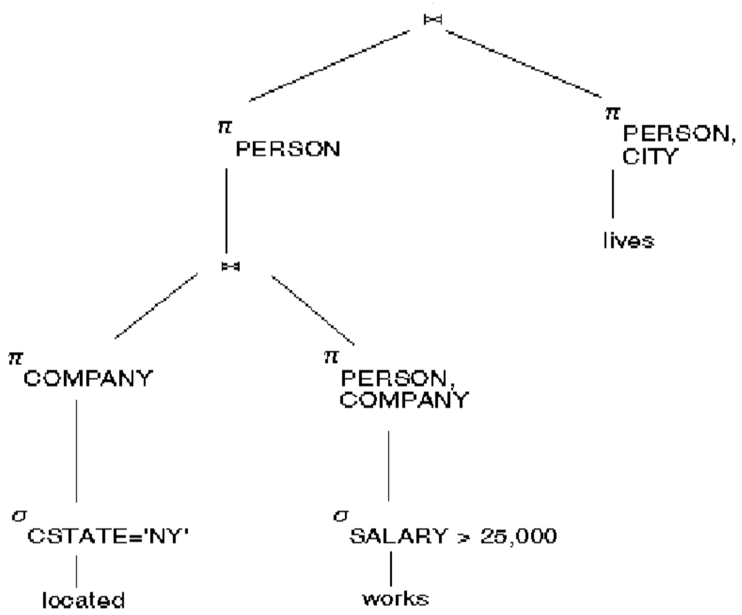


QUERY OPTIMIZATION EXAMPLE

T_2: Using commutativity & associativity properties of join, rearrange join order



T_3: Introduce projections



Estimation of Query-Processing Cost

- Databases store the following statistics for each relation:
  - n_r, the number of tuples in relation r
  - V(A, r), the number of distinct values that appear in the relation r for attribute A

- Then using the above statistics, sizes of various query expressions can be estimated as follows:
  - $| r \times s | = n_r n_s$
  - $| \sigma_A(r) | = n_r / V(A, r)$, assuming uniform value distribution
  - $| r \bowtie_\theta s | \leq | s |$, if $R \cap S$ is a key for r
  - $| r \bowtie_\theta s | = \text{MIN} \{ (n_r n_s / V(A, r)), (n_r n_s / V(A, s)) \}$, if $R \cap S = \{A\}$

STUDY PROBLEM

Consider the following query that finds the names and salaries of employees working on the 'ProductX' project:

Assume: $n_{employee} = 100$; $n_{works\_on} = 100$; $n_{project} = 10$;

Query:

$$\pi_{\text{fname, lname, salary}} (\sigma_{\text{ssn=essn \& pno=pnumber \& pname='ProductX'}} ((\text{employee} \times \text{works\_on}) \times \text{project}))$$

Give the initial query tree. Show the successive query trees generated during the query optimization process. Give a brief justification at each step.