CSE 444: Database Internals

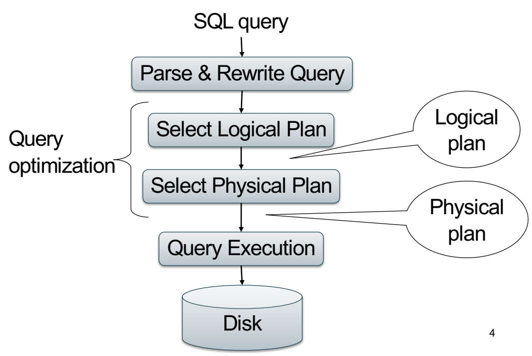
Lecture 10
Query Optimization (part 1)

Know how to compute the cost of a plan

Next: Find a good plan automatically?

This is the role of the query optimizer

Query Optimization Overview



What We Already Know…

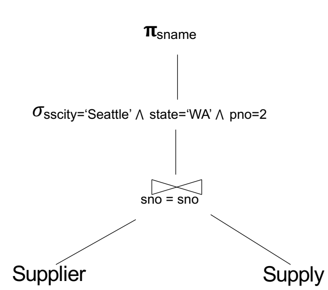
```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
```

For each SQL query….

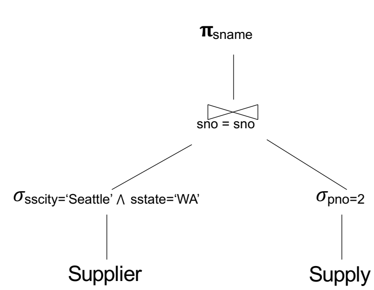
```
SELECT S.sname
FROM Supplier S, Supply U
WHERE S.scity='Seattle' AND S.sstate='WA'
AND S.sno = U.sno
AND U.pno = 2
```

There exist many logical query plans…

Example Query: Logical Plan 1



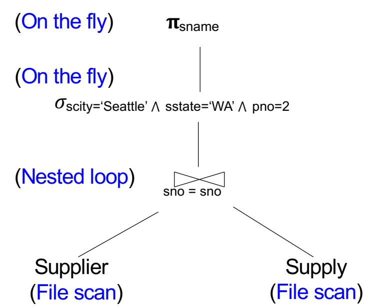
Example Query: Logical Plan 2



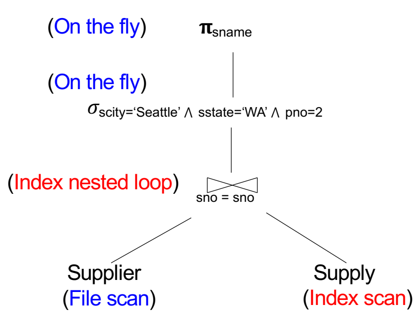
What We Also Know

- For each logical plan…
- There exist many physical plans

Example Query: Physical Plan 1



Example Query: Physical Plan 2



Query Optimizer Overview

- **Input**: A logical query plan
- **Output**: A good physical query plan

Key Decisions

Search Space

Optimization rules

Optimization algorithm

Query Optimizer Overview

- **Input**: A logical query plan
- **Output**: A good physical query plan
- **Basic query optimization algorithm**
  - Enumerate alternative plans (logical and physical)
  - Compute estimated cost of each plan
    - Compute number of I/Os
    - Optionally take into account other resources
  - Choose plan with lowest cost
  - This is called cost-based optimization

Lessons

- No magic “best” plan: depends on the data
- In order to make the right choice
  - Need to have ***statistics*** over the data
  - The B’s, the T’s, the V’s
  - Commonly: histograms over base data
    - In SimpleDB as well… see lab 5.

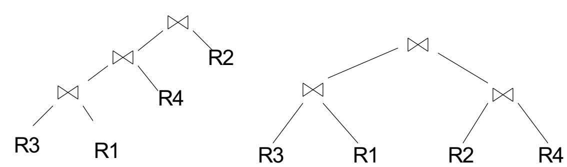
Outline

- Search space
- Algorithm for enumerating query plans

Relational Algebra Equivalences

- Selections
  - Commutative: $\sigma_{c1}(\sigma_{c2}(R))$ same as $\sigma_{c2}(\sigma_{c1}(R))$
  - Cascading: $\sigma_{c1 \wedge c2}(R)$ same as $\sigma_{c2}(\sigma_{c1}(R))$
- Projections
  - Cascading
- Joins
  - Commutative : $R \bowtie S$ same as $S \bowtie R$
  - Associative: $R \bowtie (S \bowtie T)$ same as $(R \bowtie S) \bowtie T$

Left-Deep Plans, Bushy Plans, and Linear Plans



Left-deep plan Bushy plan

Linear plan: One input to each join is a relation from disk
Can be either left or right input

Commutativity, Associativity, Distributivity

$R \cup S = S \cup R, \quad R \cup (S \cup T) = (R \cup S) \cup T$
$R \bowtie S = S \bowtie R, \quad R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$

$R \bowtie (S \cup T) = (R \bowtie S) \cup (R \bowtie T)$

Laws Involving Selection

$\sigma_{C \text{ AND } C'}(R) = \sigma_C(\sigma_{C'}(R)) = \sigma_C(R) \cap \sigma_{C'}(R)$
$\sigma_{C \text{ OR } C'}(R) = \sigma_C(R) \cup \sigma_{C'}(R)$
$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$

$\sigma_C(R - S) = \sigma_C(R) - S$
$\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$
$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$

Assuming C on attributes of R

Example: Simple Algebraic Laws

- Example: R(A, B, C, D), S(E, F, G)

$\sigma_{F=3}(R \bowtie_{D=E} S) =$

$\sigma_{A=5 \text{ AND } G=9}(R \bowtie_{D=E} S) =$

Example: Simple Algebraic Laws

- Example: R(A, B, C, D), S(E, F, G)

$\sigma_{F=3}(R \bowtie_{D=E} S) = R \bowtie_{D=E} \sigma_{F=3}(S)$

$\sigma_{A=5 \text{ AND } G=9}(R \bowtie_{D=E} S) =$

Example: Simple Algebraic Laws

- Example: R(A, B, C, D), S(E, F, G)

$\sigma_{F=3}(R \bowtie_{D=E} S) = R \bowtie_{D=E} \sigma_{F=3}(S)$

$\sigma_{A=5 \text{ AND } G=9}(R \bowtie_{D=E} S) = \sigma_{A=5}(R) \bowtie_{D=E} \sigma_{G=9}(S)$

Laws Involving Projections

$\Pi_M(R \bowtie S) = \Pi_M(\Pi_P(R) \bowtie \Pi_Q(S))$

$\Pi_M(\Pi_N(R)) = \Pi_M(R)$

/* note that $M \subseteq N$ */

- Example R(A,B,C,D), S(E, F, G)

$\Pi_{A,B,G}(R \bowtie_{D=E} S) = \Pi_{?}(\Pi_{?}(R) \bowtie_{D=E} \Pi_{?}(S))$

Laws Involving Projections

$\Pi_M(R \bowtie S) = \Pi_M(\Pi_P(R) \bowtie \Pi_Q(S))$

$\Pi_M(\Pi_N(R)) = \Pi_M(R)$

/* note that $M \subseteq N$ */

- Example R(A,B,C,D), S(E, F, G)

$\Pi_{A,B,G}(R \bowtie_{D=E} S) = \Pi_{A,B,G}(\Pi_{A,B,D}(R) \bowtie_{D=E} \Pi_{E,G}(S))$

Laws involving grouping and aggregation

$\gamma_{A,\, agg(D)}(R(A,B) \bowtie_{B=C} S(C,D)) =$
$\gamma_{A,\, agg(D)}(R(A,B) \bowtie_{B=C} (\gamma_{C,\, agg(D)} S(C,D)))$

Laws involving grouping and aggregation

$\delta(\gamma_{A,\ agg(B)}(R)) = \gamma_{A,\ agg(B)}(R)$

$\gamma_{A,\ agg(B)}(\delta(R)) = \gamma_{A,\ agg(B)}(R)$

if agg is "duplicate insensitive"

Which of the following are "duplicate insensitive" ?
sum, count, avg, min, max

Laws Involving Constraints

Foreign key

Product(<u>pid</u>, pname, price, cid)
Company(<u>cid</u>, cname, city, state)

$\Pi_{pid,\ price}(Product \bowtie_{cid=cid} Company) = \Pi_{pid,\ price}(Product)$

Search Space Challenges

- Search space is huge!
  - Many possible equivalent trees
  - Many implementations for each operator
  - Many access paths for each relation
    - File scan or index + matching selection condition
- Cannot consider ALL plans
  - Heuristics: only partial plans with "low" cost

Outline

- Search space
- Algorithm for enumerating query plans

Key Decisions

Logical plan

- What logical plans do we consider (left-deep, bushy?) *Search Space*
- Which algebraic laws do we apply, and in which context(s)? *Optimization rules*
- In what order do we explore the search space? *Optimization algorithm*

Key Decisions

Physical plan

- What physical operators to use?
- What access paths to use (file scan or index)?
- Pipeline or materialize intermediate results?

These decisions also affect the *search space*

Two Types of Optimizers

- Heuristic-based optimizers:
  - Apply greedily rules that always improve plan
    - Typically: push selections down
  - Very limited: no longer used today
- Cost-based optimizers:
  - Use a cost model to estimate the cost of each plan
  - Select the “cheapest” plan
  - We focus on cost-based optimizers

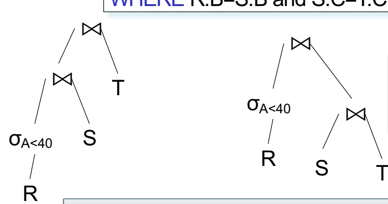
Three Approaches to Search Space Enumeration

- Complete plans
- Bottom-up plans
- Top-down plans

Complete Plans

R(A,B)
S(B,C)
T(C,D)

SELECT *
FROM R, S, T
WHERE R.B=S.B and S.C=T.C and R.A<40



Why is this search space inefficient ?

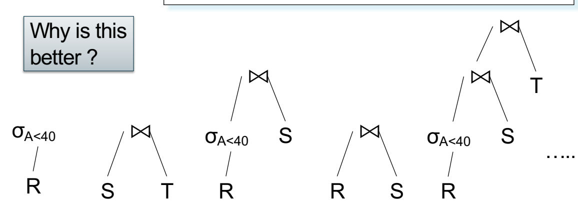
Answer: No way to do early pruning

Bottom-up Partial Plans

R(A,B)
S(B,C)
T(C,D)

SELECT *
FROM R, S, T
WHERE R.B=S.B and S.C=T.C and R.A<40

Why is this better ?

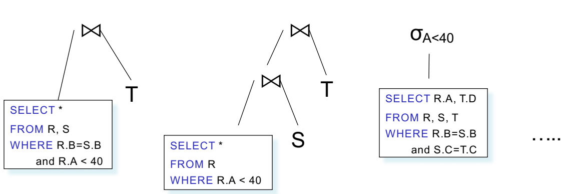


We will prune bad plans for sub-expressions

Top-down Partial Plans

R(A,B)
S(B,C)
T(C,D)

SELECT *
FROM R, S, T
WHERE R.B=S.B and S.C=T.C and R.A<40



Two Types of Plan Enumeration Algorithms

- Dynamic programming (in class)
  - Based on System R (aka Selinger) style optimizer[1979]
  - Limited to joins: *join reordering algorithm*
  - Bottom-up
- Rule-based algorithm (will not discuss)
  - Database of rules (=algebraic laws)
  - Usually: dynamic programming
  - Usually: top-down