CSE 544
Principles of Database Management Systems

Fall 2016

Lecture 8 - Query optimization

Announcements

- HW2 (SimpleDB) is due next Friday!
- Midterm in two weeks, Nov. 10, in class
- Project Milestone due on Nov. 16

Recommended Readings

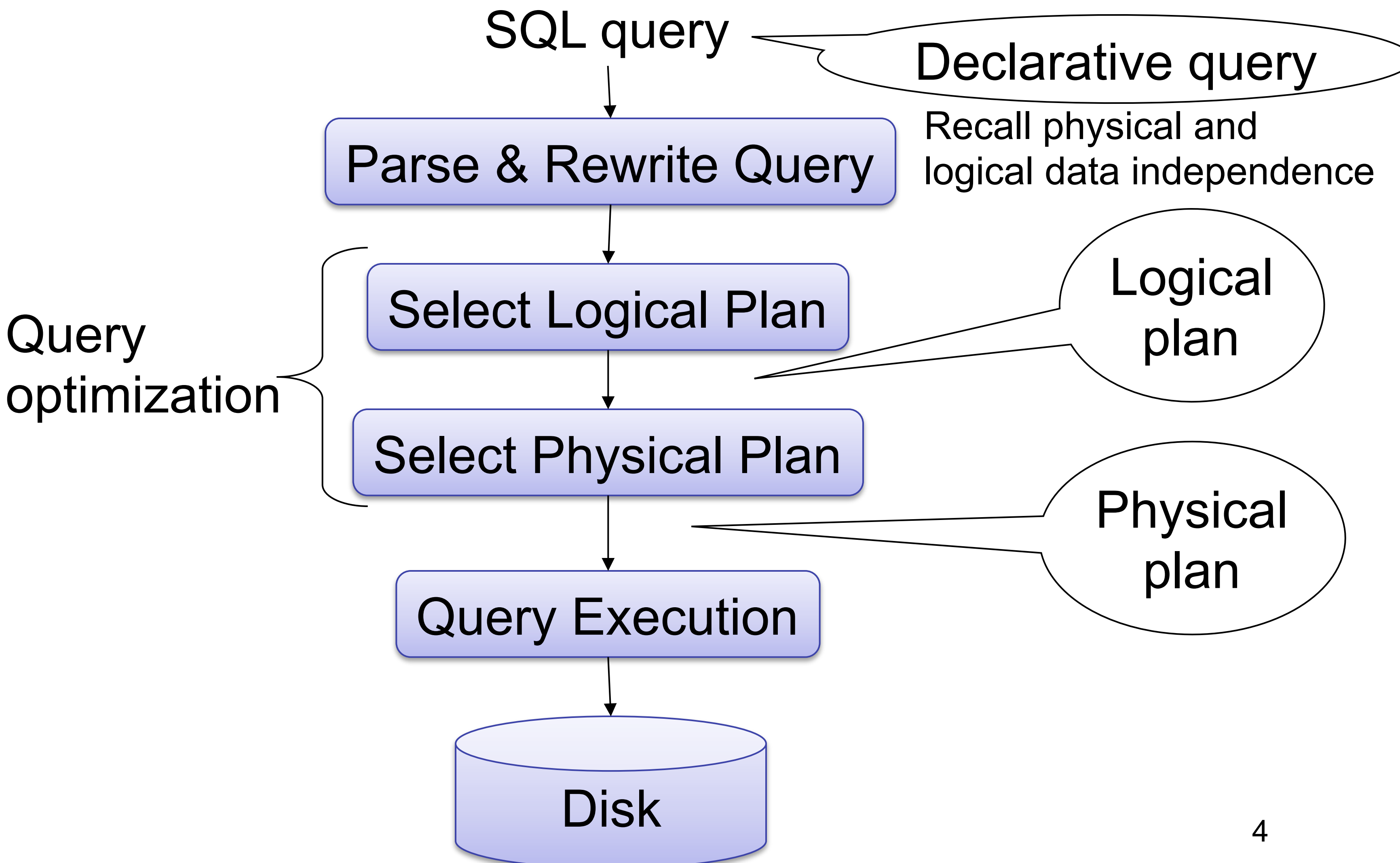
Access path selection in a relational database management system.
Selinger. et. al. SIGMOD 1979

Additional resources:

- Chaudhuri, "An Overview of Query Optimization in Relational Systems," Proceedings of ACM PODS, 1998

- Database management systems.
  Ramakrishnan and Gehrke.
  Third Ed. Chapter 15.

Query Optimization Motivation



What We Already Know…

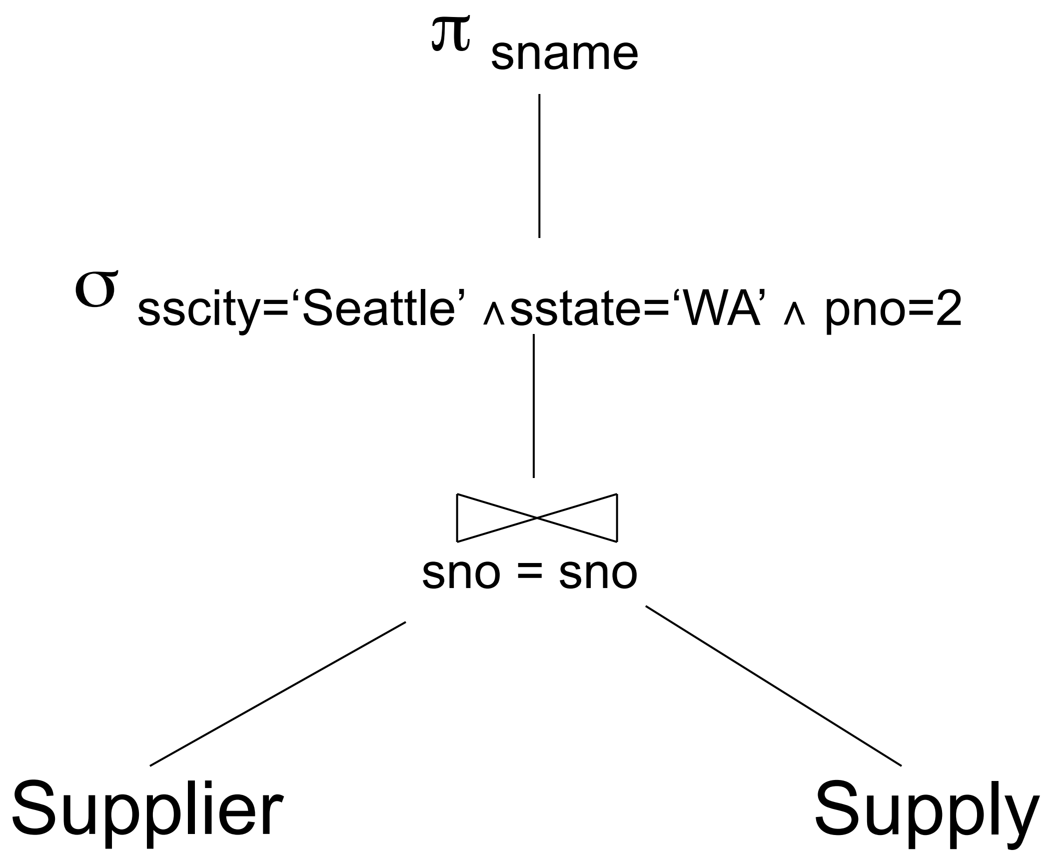
```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
```

For each SQL query….

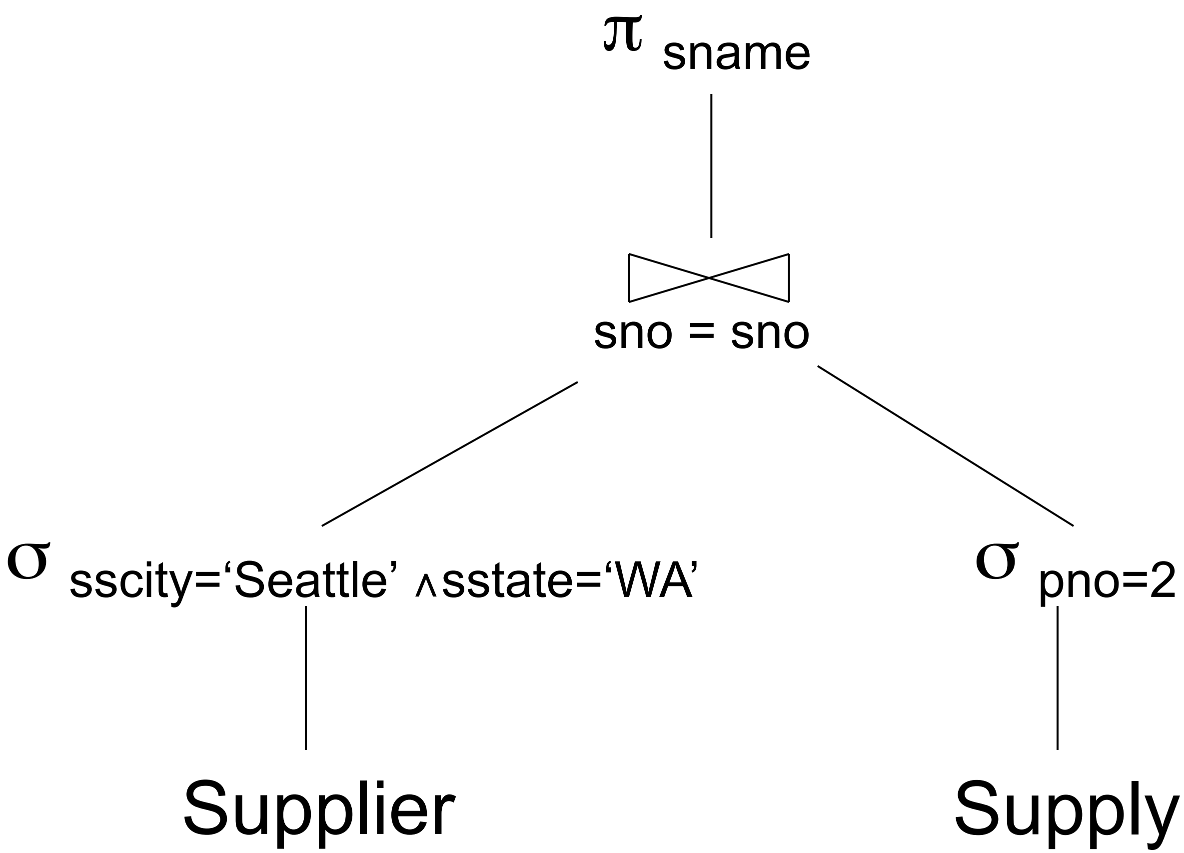
```
SELECT S.sname
FROM Supplier S, Supply U
WHERE S.scity='Seattle' AND S.sstate='WA'
AND S.sno = U.sno
AND U.pno = 2
```

There exist many logical query plan…

Example Query: Logical Plan 1



Example Query: Logical Plan 2

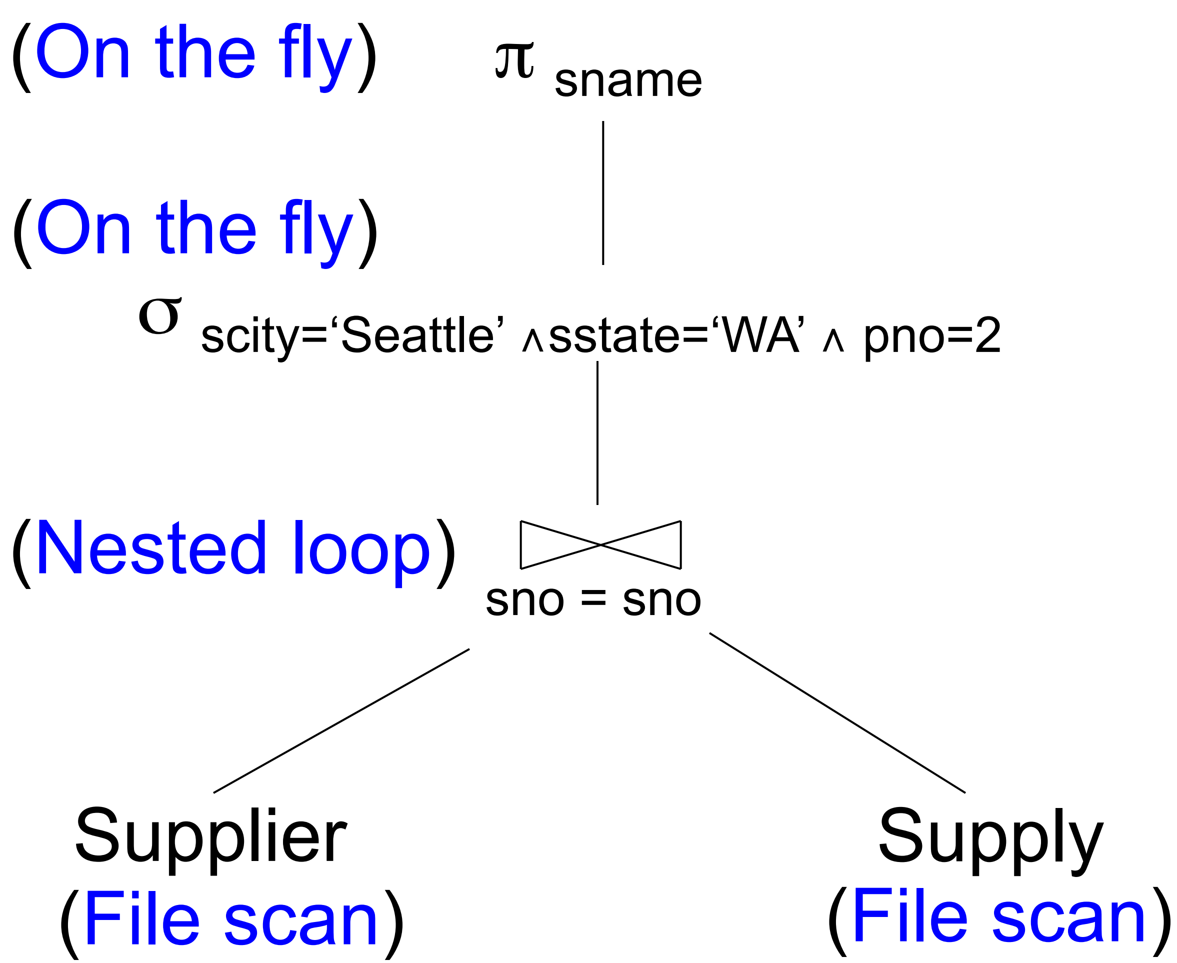


What We Also Know

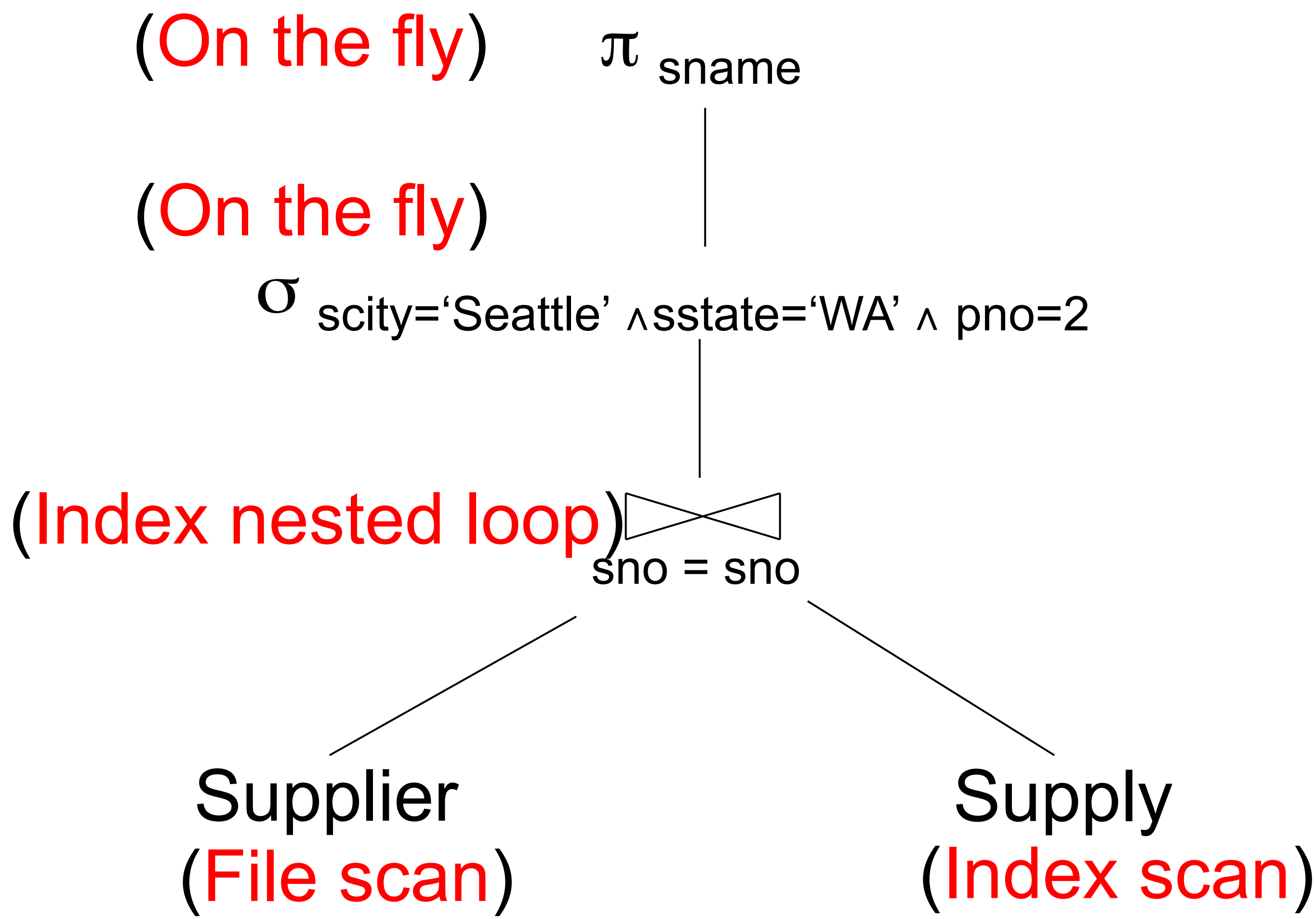
- For each logical plan…

- There exist many physical plans

Example Query: Physical Plan 1



Example Query: Physical Plan 2



Query Optimization

<u>Three major components</u>:

1. Cardinality and cost estimation

2. Search space

3. Plan enumeration algorithms

Estimating Cost of a Query Plan

Goal: compute the cost of an entire physical query plan

- We already know how to
  - Compute the cost of different operations in terms of number Ios, given the T(R)'s and the B(R)'s
- We still need to do
  - Access path selection: compute cost of retrieving tuples from disk with different access paths
  - Size estimation: compute the T(R)'s and the B(R)'s for intermediate relations R

Access Path

Access path: a way to retrieve tuples from a table

- A file scan
- An index *plus* a matching selection condition

Access Path Selection

- Supplier(sid,sname,scity,sstate)
- Selection condition: sid > 300 ∧ scity='Seattle'
- Indexes: B+-tree on sid and B+-tree on scity
- Which access path should we use?
- We should pick the **most selective** access path

Access Path Selectivity

- **Access path selectivity is the number of pages retrieved if we use this access path**
  - Most selective retrieves fewest pages

- As we saw earlier, **for equality predicates**
  - Selection on equality: $\sigma_{a=v}(R)$
  - V(R, a) = # of distinct values of attribute a
  - 1/V(R,a) is thus the reduction factor
  - Clustered index on a: cost B(R)/V(R,a)
  - Unclustered index on a: cost T(R)/V(R,a)
  - (we are ignoring I/O cost of index pages for simplicity)

Selectivity for Range Predicates

Selection on range: $\sigma_{a>v}(R)$

- How to compute the selectivity?
- Assume values are uniformly distributed
- Reduction factor X
- X = (Max(R,a) - v) / (Max(R,a) - Min(R,a))

- Clustered index on a: cost B(R)*X
- Unclustered index on a: cost T(R)*X

Back to Our Example

- Selection condition: **sid > 300 $\wedge$ scity='Seattle'**
  - Index I1: B+-tree on sid clustered
  - Index I2: B+-tree on scity unclustered

- Let's assume
  - V(Supplier,scity) = 20
  - Max(Supplier, sid) = 1000, Min(Supplier,sid)=1
  - B(Supplier) = 100, T(Supplier) = 1000

- **Cost I1: B(R) * (Max-v)/(Max-Min) = 100*700/999 $\approx$ 70**
- **Cost I2: T(R) * 1/V(Supplier,scity) = 1000/20 = 50**

Selectivity with Multiple Conditions

What if we have an index on multiple attributes?

- Example selection $\sigma_{a=v1 \wedge b=v2}(R)$ and index on <a,b>

How to compute the selectivity?

- Assume attributes are independent
- $X = 1 / (V(R,a) * V(R,b))$

- Clustered index on <a,b>: cost $B(R)*X$
- Unclustered index on <a,b>: cost $T(R)*X$

Estimating Cost of a Query Plan

Goal: compute the cost of an entire physical query plan

- We already know how to
  - Compute the cost of different operations in terms of number Ios, given the T(R)'s and the B(R)'s
- We still need to do
  - Access path selection: compute cost of retrieving tuples from disk with different access paths
  - Size estimation: compute the T(R)'s and the B(R)'s for intermediate relations R

Statistics on Base Data

- Collected information for each relation
  - Number of tuples (cardinality) T(R)
  - Number of physical pages B(R), clustering info
  - Indexes, number of keys in the index V(R,a)
  - Statistical information on attributes
    - Min value, max value, number distinct values
    - Histograms
  - Correlations between columns (hard)

- Collection approach: periodic, using sampling

Size Estimation

Projection: output size same as input size

Selection: multiply input size by reduction factor

- Similar to what we did for estimating access path selectivity
- Assume independence between conditions in the predicate
- Examples:

$$T(\sigma_{A=\ldots}(R)) = T(R) / V(R,A)$$
$$T(\sigma_{A=\ldots \wedge B=\ldots}(R)) = T(R) / (V(R,A) * V(R,B))$$

Estimating Result Sizes

Join R $\bowtie$ S

- Take product of cardinalities of relations R and S
- Apply reduction factors for each term in join condition
- Terms are of the form: column1 = column2
- Reduction: 1/ (MAX(V(R,column1), V(S,column2))
- Why? Will explain next...

Assumptions

- *Containment of values*: if V(R,A) <= V(S,B), then the set of A values of R is included in the set of B values of S
  - Note: this indeed holds when A is a foreign key in R, and B is a key in S

- *Preservation of values*: for any other attribute C, $V(R \bowtie_{A=B} S, C) = V(R, C)$ (or V(S, C))

Selectivity of $R \bowtie_{A=B} S$

Assume $V(R,A) <= V(S,B)$

- Each tuple t in R joins with $T(S)/V(S,B)$ tuple(s) in S

- Hence $T(R \bowtie_{A=B} S) = T(R)\ T(S) / V(S,B)$

In general: $T(R \bowtie_{A=B} S) = T(R)\ T(S) / \max(V(R,A),V(S,B))$

Complete Example

Supplier(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, quantity)

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and  y.pno = 2
   and x.scity = 'Seattle'
   and x.sstate = 'WA'
```

- Some statistics
  - T(Supplier) = 1000 records
  - T(Supply) = 10,000 records
  - B(Supplier) = 100 pages
  - B(Supply) = 100 pages
  - V(Supplier,scity) = 20, V(Suppliers,state) = 10
  - V(Supply,pno) = 2,500
  - Both relations are clustered
- M = 11

Computing the Cost of a Plan

- Estimate cardinality in a bottom-up fashion
  - Cardinality is the size of a relation (nb of tuples)
  - Compute size of *all* intermediate relations in plan

- Estimate cost by using the estimated cardinalities

T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 11

Physical Query Plan 1

(On the fly) π_{sname}

Selection and project on-the-fly
-> No additional cost.

(On the fly) $\sigma_{\text{scity='Seattle'} \wedge \text{sstate='WA'} \wedge \text{pno=2}}$

(Nested loop) $\bowtie_{\text{sno = sno}}$

Total cost of plan is thus cost of join:
= B(Supplier)+B(Supplier)*B(Supplies)
= 100 + 100 * 100
= 10,100 I/Os

Supplier (File scan)

Supply (File scan)

T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 11

Physical Query Plan 2

(On the fly) π_{sname} (4)

(Sort-merge join) $\bowtie_{\text{sno = sno}}$ (3)

(Scan write to T1) (1) $\sigma_{\text{scity='Seattle'} \wedge \text{sstate='WA'}}$

(Scan write to T2) (2) $\sigma_{\text{pno=2}}$

Supplier (File scan)

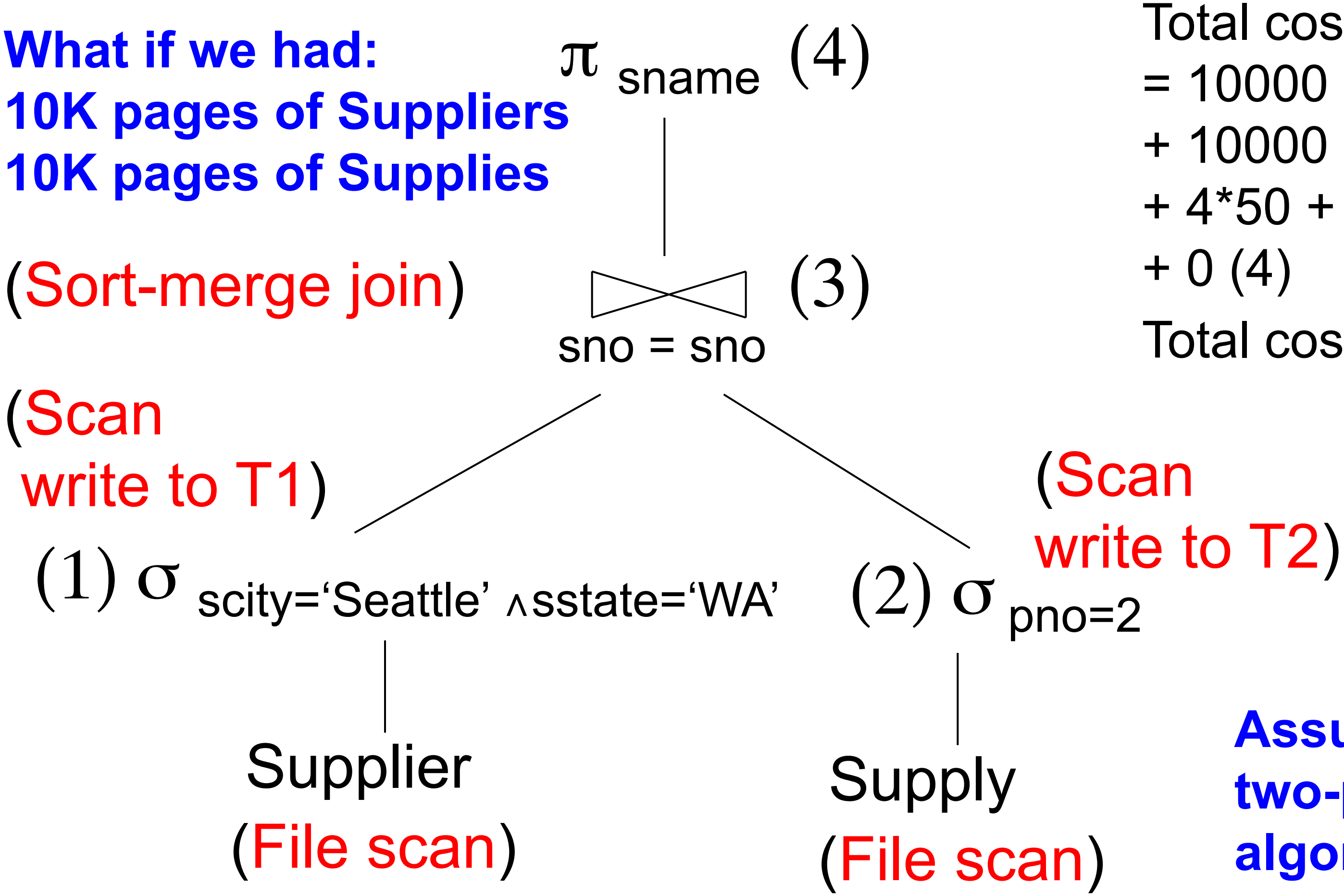
Supply (File scan)

Total cost
= 100 + 100 * 1/20 * 1/10 (1)
+ 100 + 100 * 1/2500 (2)
+ 2 (3)
+ 0 (4)

Total cost ≈ **204 I/Os**

Plan 2 with Different Numbers

What if we had:
10K pages of Suppliers
10K pages of Supplies



Total cost
= 10000 + 50 (1)
+ 10000 + 4 (2)
+ 4*50 + 2*4 + 4 + 50 (3)
+ 0 (4)

Total cost ≈ 20,316 I/Os

Assuming naive two-pass sort algorithm

T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 11

Physical Query Plan 3

(On the fly) (4) π_{sname}

(On the fly)

(3) $\sigma_{\text{scity='Seattle'} \wedge \text{sstate='WA'}}$

(2) $\bowtie_{\text{sno = sno}}$ (Index nested loop)

4 tuples

(Use hash index)

(1) $\sigma_{\text{pno=2}}$

Supply

(Hash index on pno)
Assume: clustered

Supplier

(Hash index on sno)
Clustering does not matter

Total cost
= 1 (1)
+ 4 (2)
+ 0 (3)
+ 0 (3)

Total cost ≈ **5 I/Os**

Simplifications

- In the previous examples, we assumed that all index pages were in memory
- When this is not the case, we need to add the cost of fetching index pages from disk

Different Cost Models

- In previous examples, we considered IO costs
- Typically, want IO+CPU
- For parallel/distributed queries, add network bandwidth
- If need to compare *logical* plans
  - Compute the cardinality of each *intermediate* relation
  - Sum up all the cardinalities

Summary

Goal: compute the cost of an entire physical query plan

- We already know how to
  - Compute the cost of different operations in terms of number Ios, given the T(R)’s and the B(R)’s
- We still need to do
  - Access path selection: compute cost of retrieving tuples from disk with different access paths
  - Size estimation: compute the T(R)’s and the B(R)’s for intermediate relations R

Query Optimization

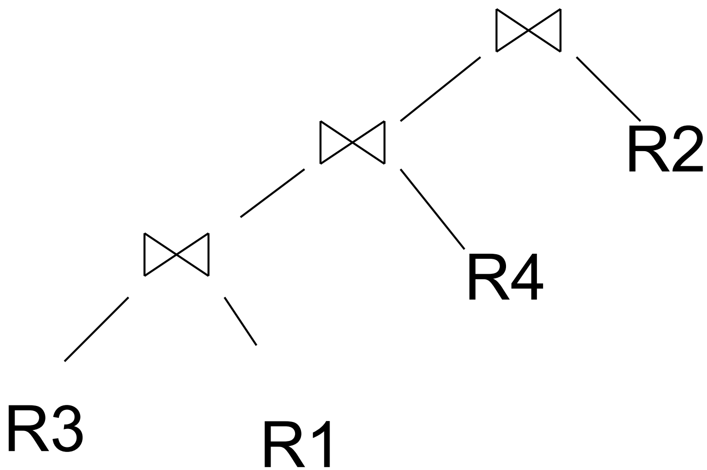
Three major components:

1. Cardinality and cost estimation
2. Search space
3. Plan enumeration algorithms

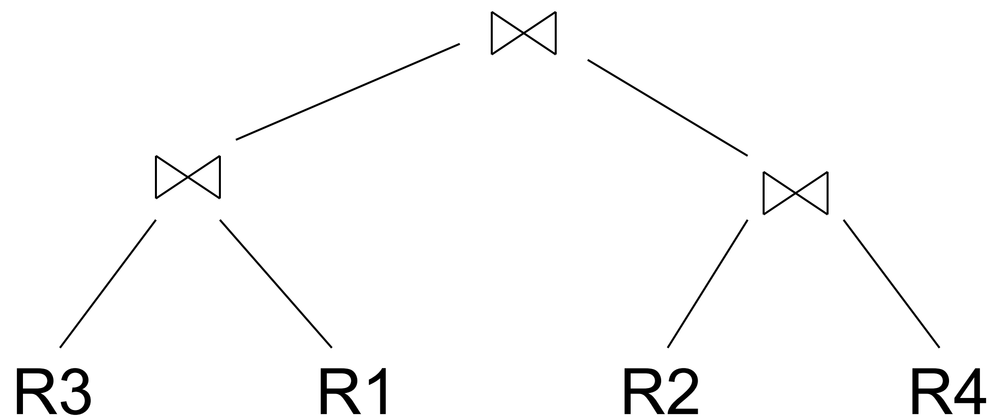
Relational Algebra Laws

- 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 and Bushy Plans



Left-deep plan



Bushy plan

Relational Algebra Laws

- Selects, projects, and joins
  - We can commute and combine all three types of operators
  - We just have to be careful that the fields we need are available when we apply the operator
  - Relatively straightforward. See book 15.3.

- More info in optional paper (by Chaudhuri), Section 4.

Group-by and Join

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

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

Group-by and Join

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

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

These are very powerful laws.
They were introduced only in the 90's.

Search Space Challenges

- Search space is huge!
  - Many possible equivalent trees (logical)
  - Many implementations for each operator (physical)
  - Many access paths for each relation (physical)

- Cannot consider ALL plans
- Want a search space that includes low-cost plans

- Typical compromises:
  - Only left-deep plans
  - Only plans without cartesian products
  - Always push selections down to the leaves

Query Optimization

Three major components:

1. Cardinality and cost estimation
2. Search space
3. Plan enumeration algorithms

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

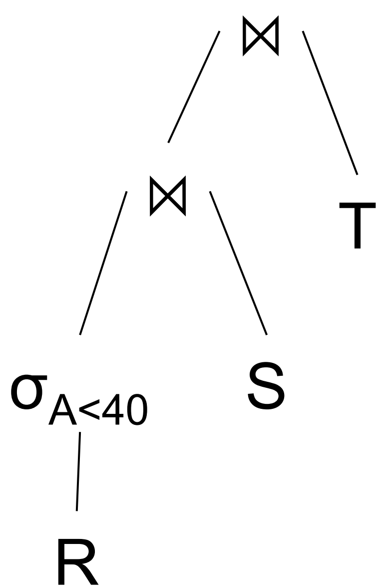
Three Approaches to Search Space Enumeration

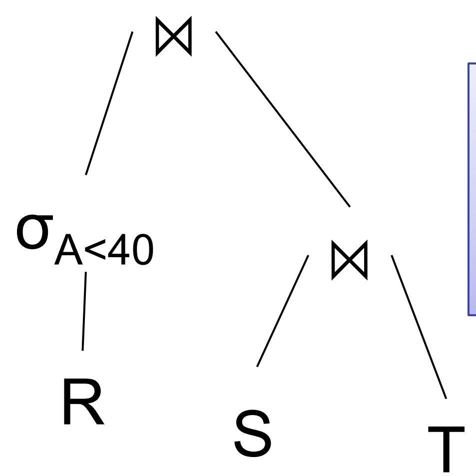
- Complete plans
- Bottom-up plans
- Top-down plans

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
```





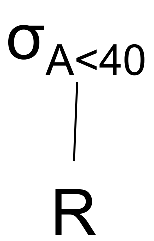
Why is this search space inefficient ?

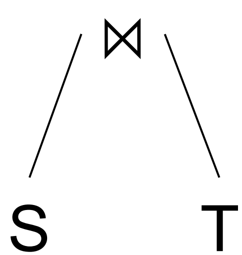
Bottom-up Partial Plans

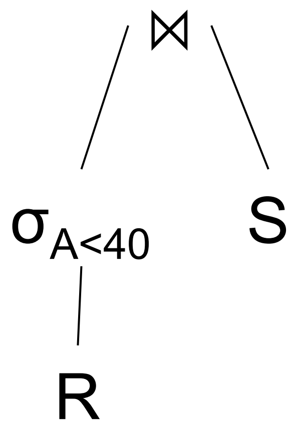
R(A,B)
S(B,C)
T(C,D)

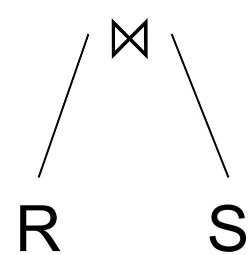
```
SELECT *
FROM R, S, T
WHERE R.B=S.B and S.C=T.C and R.A<40
```

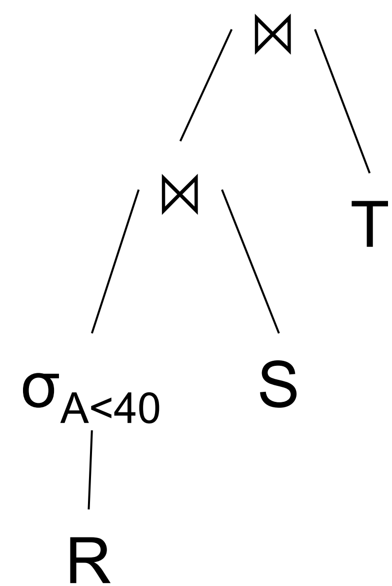
Why is this better ?









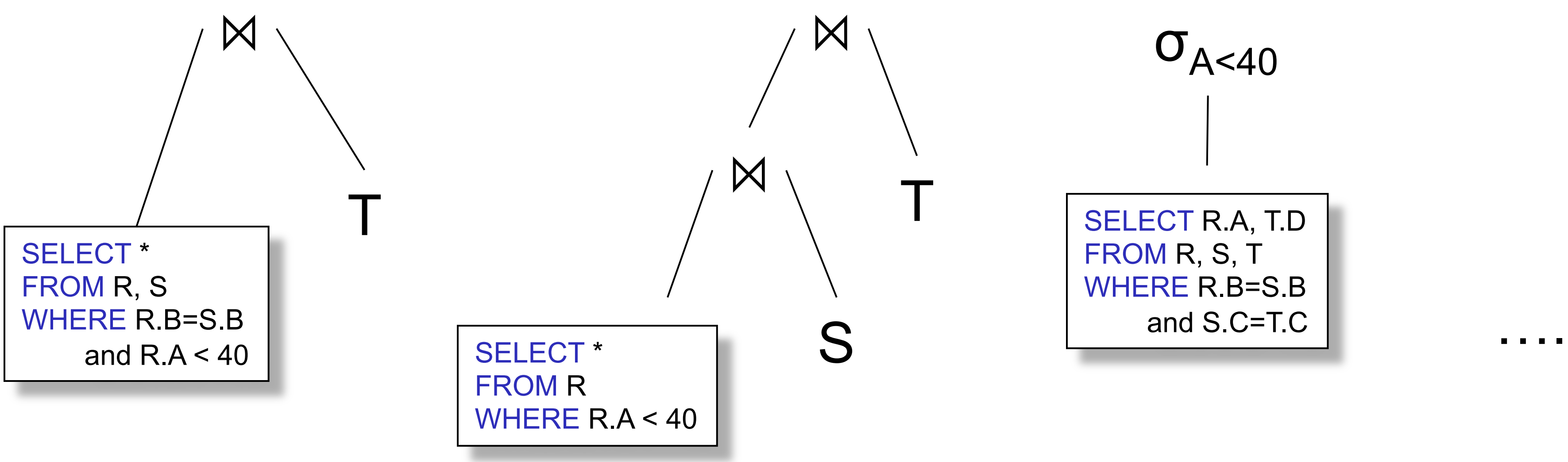


.....

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

System R Search Space

- Only left-deep plans
  - Enable dynamic programming for enumeration
  - Facilitate tuple pipelining from outer relation
- Consider plans with all "interesting orders"
- Perform cross-products after all other joins (heuristic)
- Only consider nested loop & sort-merge joins
- Consider both file scan and indexes
- Try to evaluate predicates early

Plan Enumeration Algorithm

- Idea: use dynamic programming
- For each subset of {R1, …, Rn}, compute the best plan for that subset
- In increasing order of set cardinality:
  - Step 1: for {R1}, {R2}, …, {Rn}
  - Step 2: for {R1,R2}, {R1,R3}, …, {Rn-1, Rn}
  - …
  - Step n: for {R1, …, Rn}
- It is a bottom-up strategy
- A subset of {R1, …, Rn} is also called a *subquery*

Dynamic Programming Algo.

- For each subquery $Q \subseteq \{R1, \ldots, Rn\}$ compute the following:
  - Size(Q)
  - A best plan for Q: Plan(Q)
  - The cost of that plan: Cost(Q)

Dynamic Programming Algo.

- **Step 1**: Enumerate all single-relation plans
  - Consider selections on attributes of relation
  - Consider all possible access paths
  - Consider attributes that are not needed
  - Compute cost for each plan
  - Keep cheapest plan per "interesting" output order

Dynamic Programming Algo.

- **Step 2**: Generate all two-relation plans
  - For each each single-relation plan from step 1
  - Consider that plan as outer relation
  - Consider every other relation as inner relation
  - Compute cost for each plan
  - Keep cheapest plan per “interesting” output order

Dynamic Programming Algo.

- **Step 3**: Generate all three-relation plans
  - For each each two-relation plan from step 2
  - Consider that plan as outer relation
  - Consider every other relation as inner relation
  - Compute cost for each plan
  - Keep cheapest plan per “interesting” output order
- **Steps 4 through n**: repeat until plan contains all the relations in the query

Commercial Query Optimizers

DB2, Informix, Microsoft SQL Server, Oracle 8

- Inspired by System R
  - Left-deep plans and dynamic programming
  - Cost-based optimization (CPU and IO)

- Go beyond System R style of optimization
  - Also consider right-deep and bushy plans (e.g., Oracle and DB2)
  - Variety of additional strategies for generating plans (e.g., DB2 and SQL Server)

Other Query Optimizers

- Randomized plan generation
  - Genetic algorithm
  - PostgreSQL uses it for queries with many joins

- Rule-based
  - ***Extensible*** collection of rules
  - Rule = Algebraic law with a direction
  - Algorithm for firing these rules
    - Generate many alternative plans, in some order
    - Prune by cost
  - Startburst (later DB2) and Volcano (later SQL Server)