CSE 544
Principles of Database Management Systems

Magdalena Balazinska

Fall 2007

Lecture 9 - Query optimization

References

- **Access path selection in a relational database management system.**

  Selinger. et. al. SIMOD 1979

- **Database management systems.**

  Ramakrishnan and Gehrke.

  Third Ed. **Chapter 15.**

Outline

- **Basic query optimization algorithm**

- **Typical query optimizer (based on System R)**
  - Estimating the cost of a query plan
  - Search space
  - Algorithm for enumerating query plans

- **Other types of optimizers**

Query Optimization Algorithm

- For a query
  - There exists many physical query plans
  - Query optimizer needs to pick a good one

- Basic query optimization algorithm
  - Enumerate alternative plans
  - 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

Outline

- **Basic query optimization algorithm**
- **Typical query optimizer (based on System R)**
  - Estimating the cost of a query plan
  - Search space
  - Algorithm for enumerating query plans
- **Other types of optimizers**

Estimating Cost of a Query Plan

- We already how to
  - Compute the cost of different operations

- We still need to
  - Compute cost of retrieving tuples from disk with different access paths (for more sophisticated predicates than equality)
  - Compute cost of a complete plan

Access Path

- **Access path**: a way to retrieve tuples from a table
  - A file scan
  - An index *plus* a matching selection condition

- Index matches selection condition if it can be used to retrieve just tuples that satisfy the condition
  - Example: Supplier(sid,sname,scity,sstate)
  - B+-tree index on (scity,sstate)
    - matches scity='Seattle'
    - does not match sid=3, does not match sstate='WA'

Access Path Selection

- Supplier(sid,sname,scity,sstate)
- Selection condition: $sid > 300 \wedge scity = \text{'Seattle'}$
- 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 ∧ 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 ≈ 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

Back to Estimating Cost of a Query Plan

- We already how to
  - Compute the cost of different operations
  - Compute cost of retrieving tuples from disk with different access paths (for more sophisticated predicates than equality)

- We still need to
  - Compute cost of a complete plan

Computing the Cost of a Plan

- Collect statistical summaries of stored data
- Compute cost in a bottom-up fashion
- For each operator compute
  - Estimate cost of executing the operation
  - Estimate statistical summary of the output data

Statistics on Base Data

- Collected information for each relation
  - Number of tuples (cardinality)
  - Indexes, number of keys in the index
  - Number of physical pages, clustering info
  - Statistical information on attributes
    - Min value, max value, number distinct values
    - Histograms
  - Correlations between columns (hard)
- Collection approach: periodic, using sampling

Computing Cost of an Operator

- The cost of executing an operator depends
  - On the operator implementation
  - On the input data
- We learned how to compute this cost last two lectures

Statistics on the Output Data

- Most important piece of information
  - **Size of operator result**
  - I.e., the number of output tuples

- 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
  - (use product of the reduction factors for the terms)

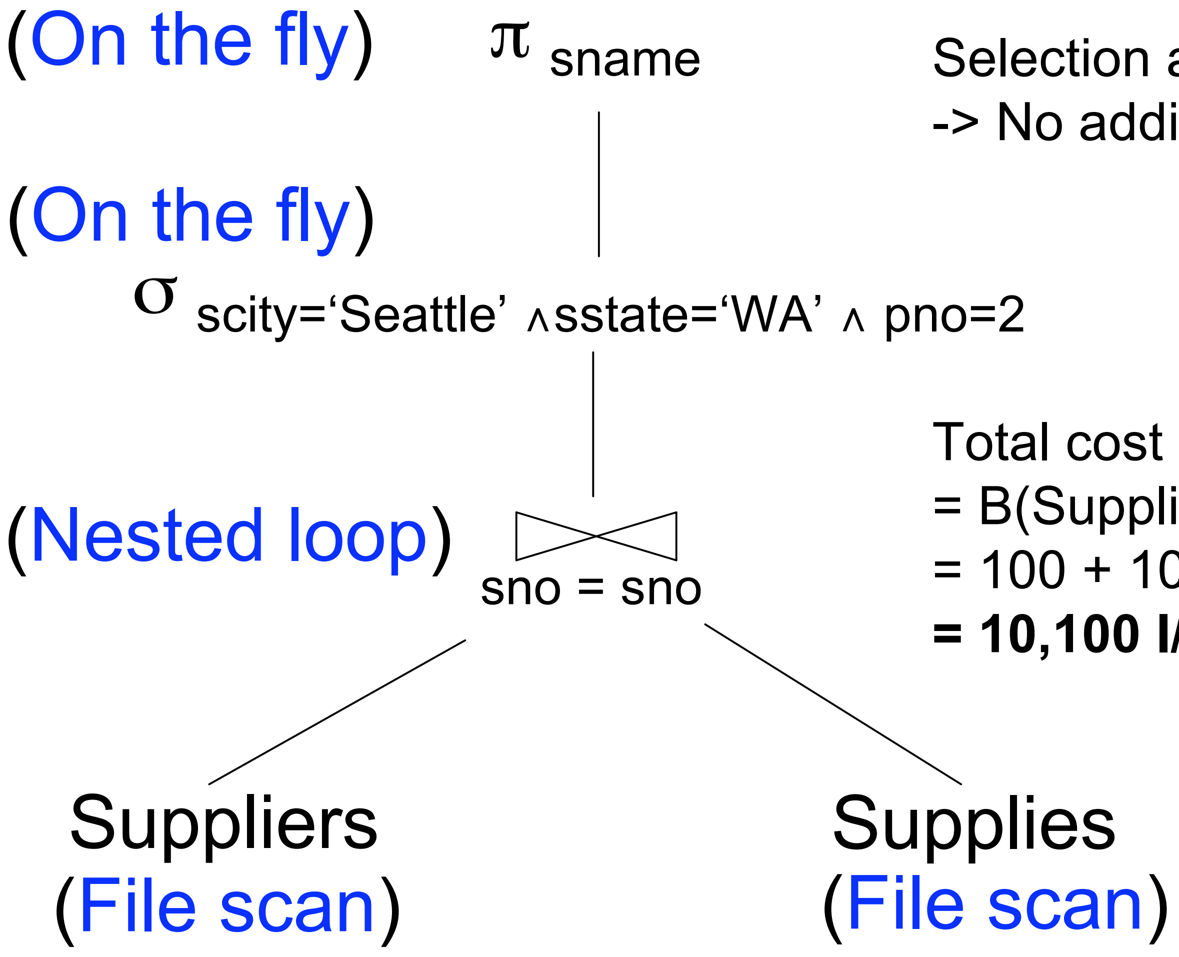
Estimating Result Sizes

- For joins 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))
  - **Assumes each value in smaller set has a matching value in the larger set**

Our Example

- Suppliers(sid,sname,scity,sstate)
- Supplies(pno,sid,quantity)
- Some statistics
  - T(Supplier) = 1000 records
  - B(Supplier) = 100 pages
  - T(Supplies) = 10,000 records
  - B(Supplies) = 100 pages
  - V(Supplier,scity) = 20, V(Supplier,state) = 10
  - V(Supplies,pno) = 3,000
  - Both relations are clustered

Physical Query Plan 1



(On the fly) π_{sname}

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

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

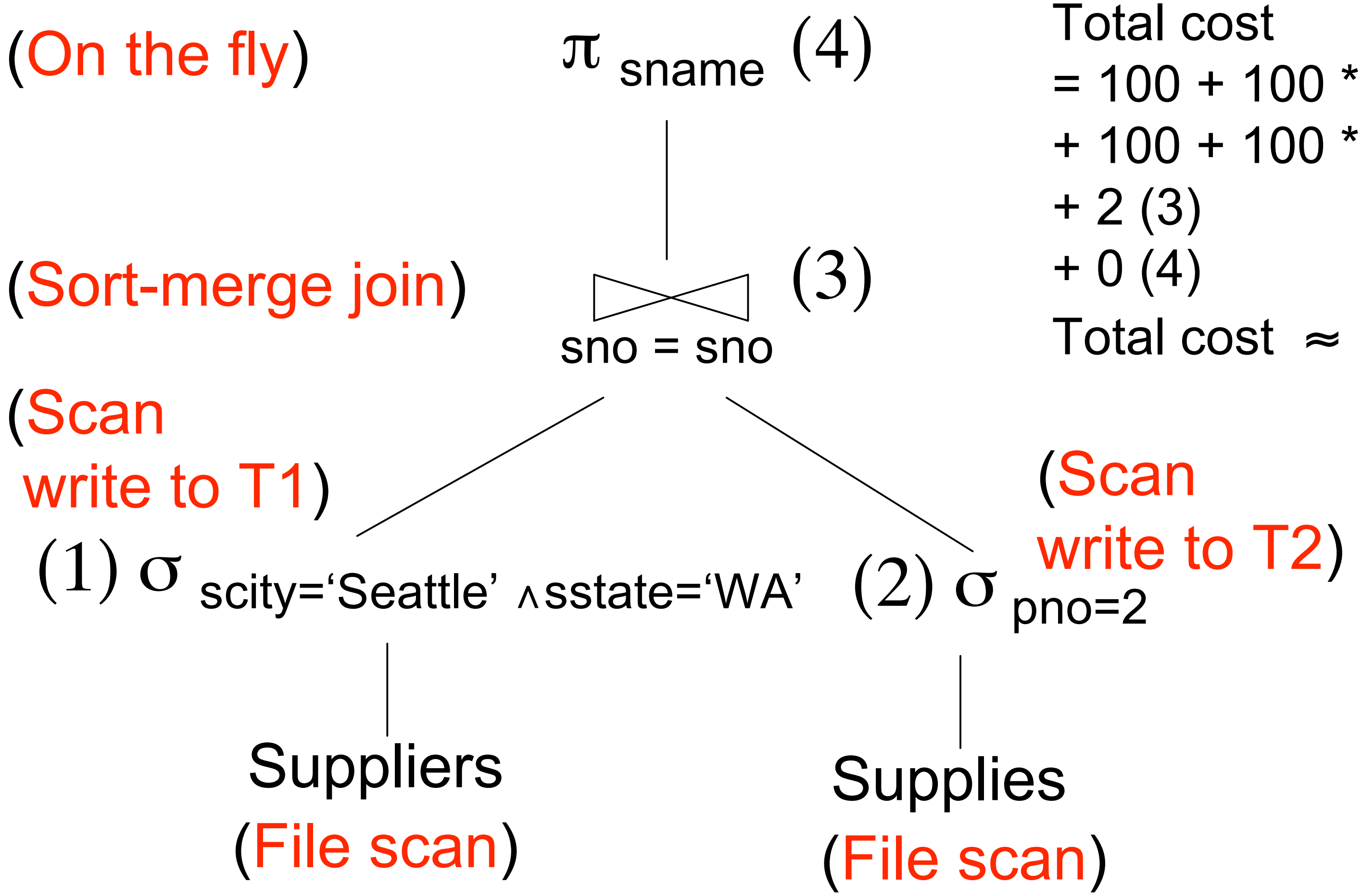
Suppliers
(File scan)

Supplies
(File scan)

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

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

Physical Query Plan 2



(On the fly) π_{sname} (4)

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

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

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

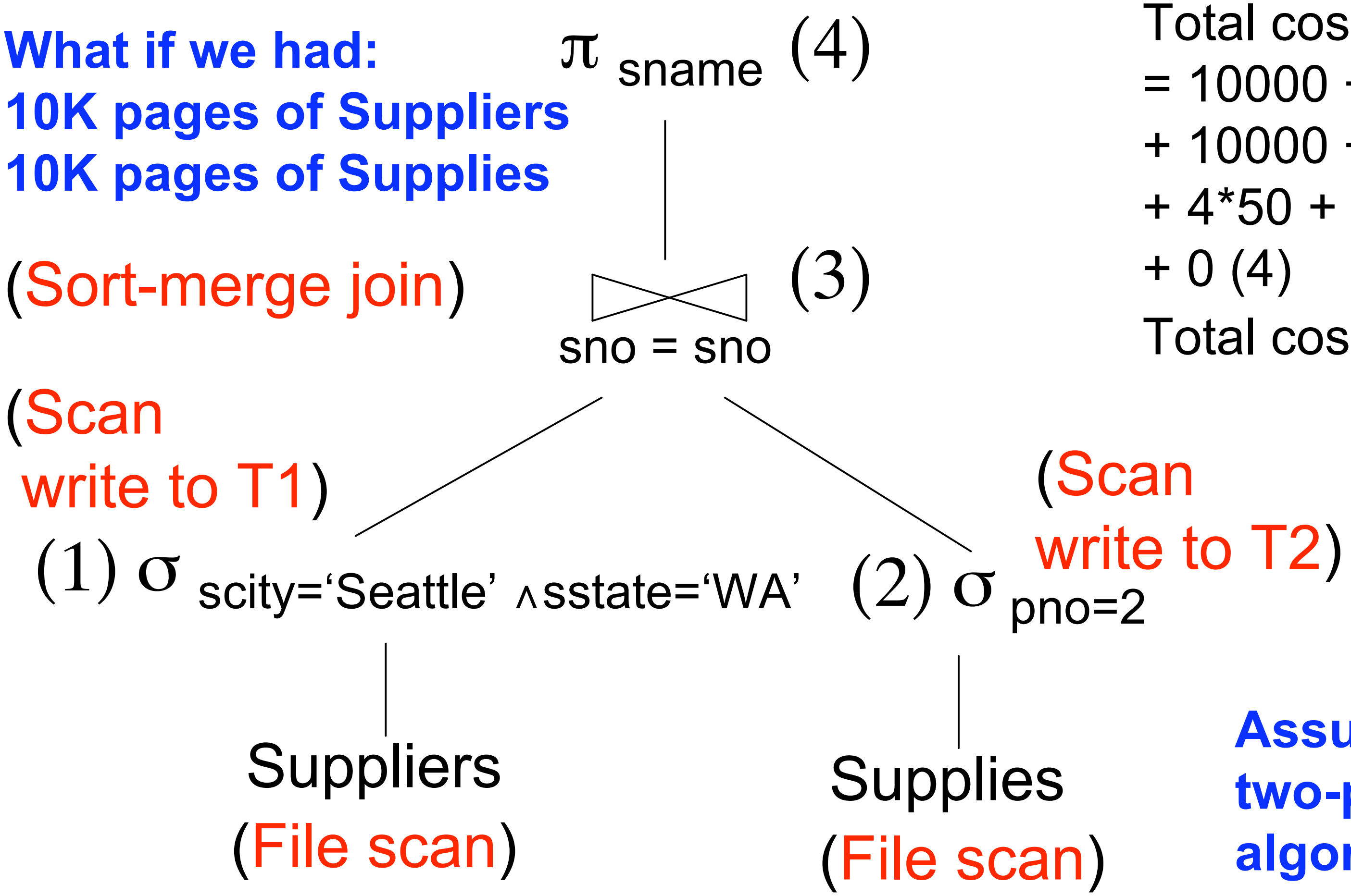
Suppliers (File scan)

Supplies (File scan)

Total cost
= 100 + 100 * 1/20 * 1/10 (1)
+ 100 + 100 * 1/3000 (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



π sname (4)

(Sort-merge join)

$\bowtie$ (3)
sno = sno

(Scan
write to T1)

(1) σ scity='Seattle' $\wedge$sstate='WA'

(2) σ pno=2

(Scan
write to T2)

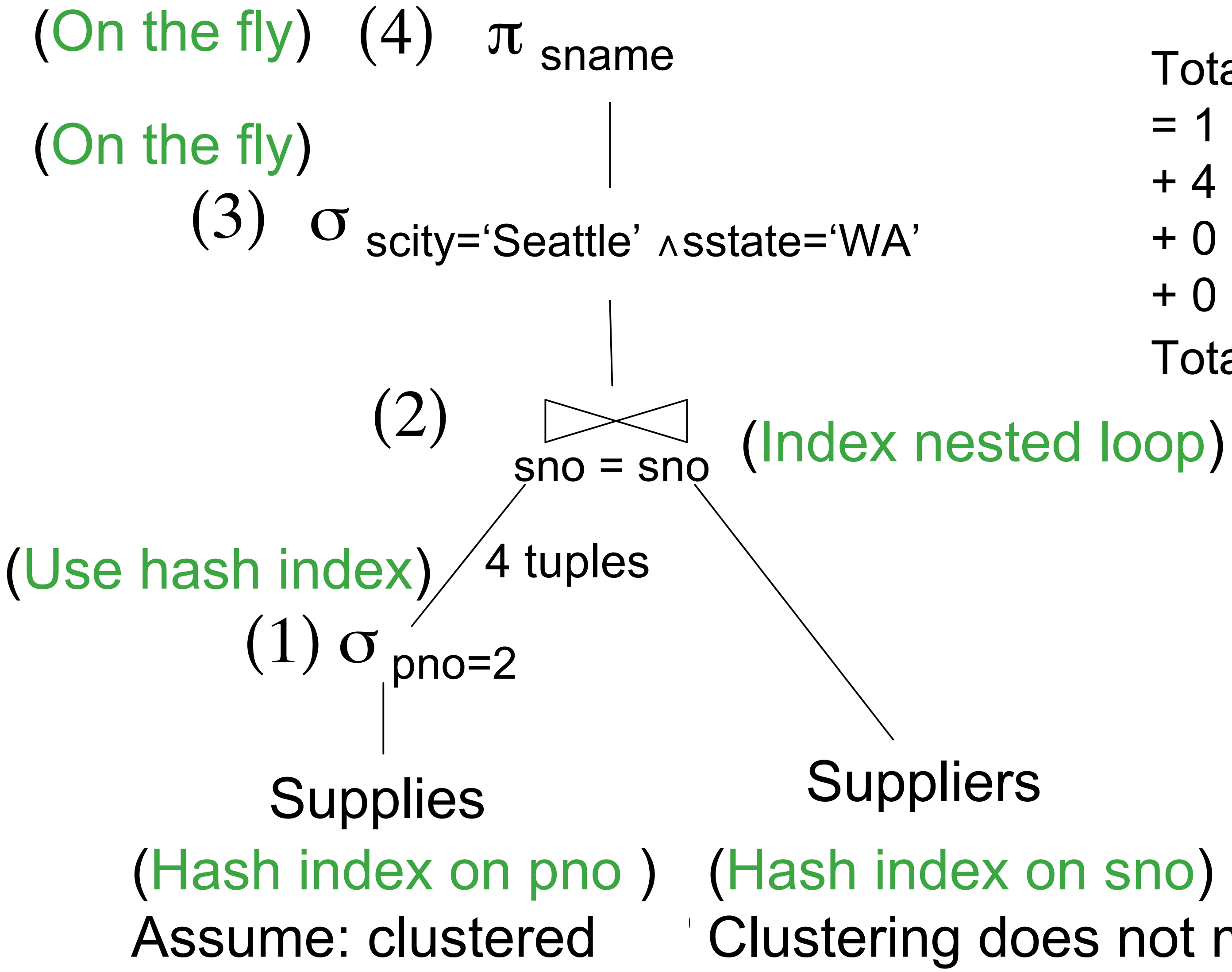
Suppliers
(File scan)

Supplies
(File scan)

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

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)

(Use hash index) 4 tuples

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

Supplies

(Hash index on pno)

Assume: clustered

Suppliers

(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 (see lecture 6)

Summary

- What we know
  - Different types of physical query plans
  - How to compute the cost of a query plan
  - Although it is hard to compute the cost accurately

- We can now compare query plans

- Let’s now consider how the query optimizer searches through the space of possible plans

Outline

- **Basic query optimization algorithm**

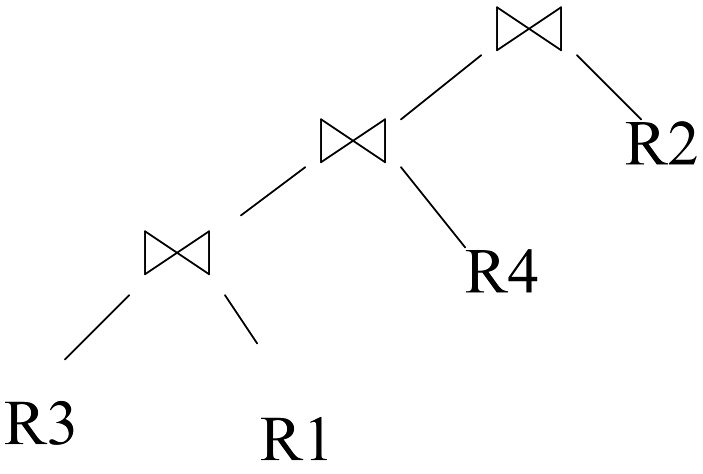
- **Typical query optimizer (based on System R)**
  - Estimating the cost of a query plan
  - Search space
  - Algorithm for enumerating query plans

- **Other types of optimizers**

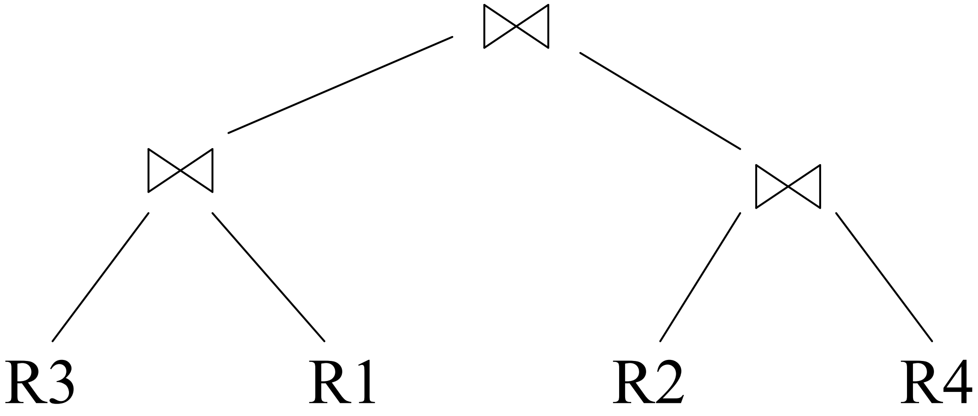
Relational Algebra Equivalences

- 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 and Bushy Plans



Left-deep plan



Bushy plan

Relational Algebra Equivalences

- 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.

- If you like this topic, more info in optional paper (by Chaudhuri), Section 4.

Search Space Challenges

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

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

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 ⊆{R1, …, 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)