




Spring 2017



# QUERY OPTIMIZATION

## [CH 15]

# Example

```
SELECT distinct ename FROM Emp E, Dept D
WHERE E.did = D.did and D.dname = 'Toy'
```

EMP (ssn, , , addr, sal, )  
10,000 employees  
1,000 pages

DEPT (, , dname, floor, mgr)  
500 departments  
50 pages

Query:  $\Pi_{\text{ename}} \sigma_{\text{dname} = \text{'Toy'}} (\text{EMP} \bowtie \text{DEPT})$

$\rho (R, \Pi_{\text{ename}} T3)$





$\rho (T3, \sigma_{\text{dname} = \text{'Toy'}} T2)$



$\rho (T2, \sigma_{\text{EMP.did} = \text{DEPT.did}} T1)$

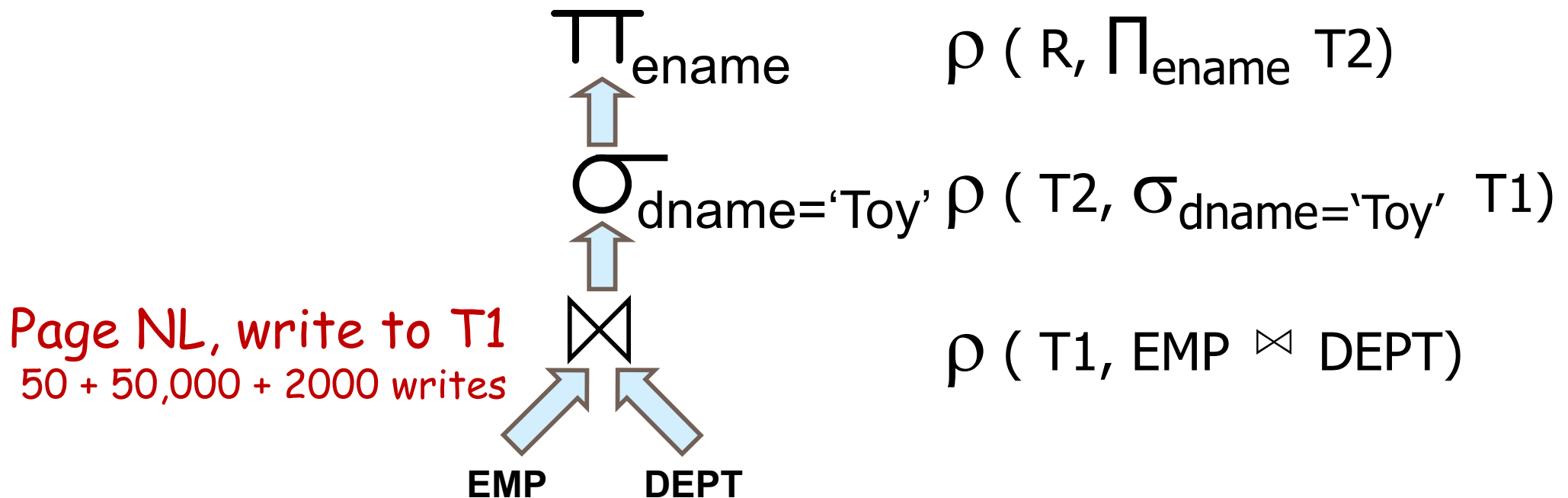
$\rho (T1, \text{EMP} \times \text{DEPT})$

50 + 50,000 + 1,000,000 writes  
(5 tuples per page in T1)

# Example

  EMP (ssn,  ename, addr, sal,  did)  
 10,000 employees  
 1,000 pages

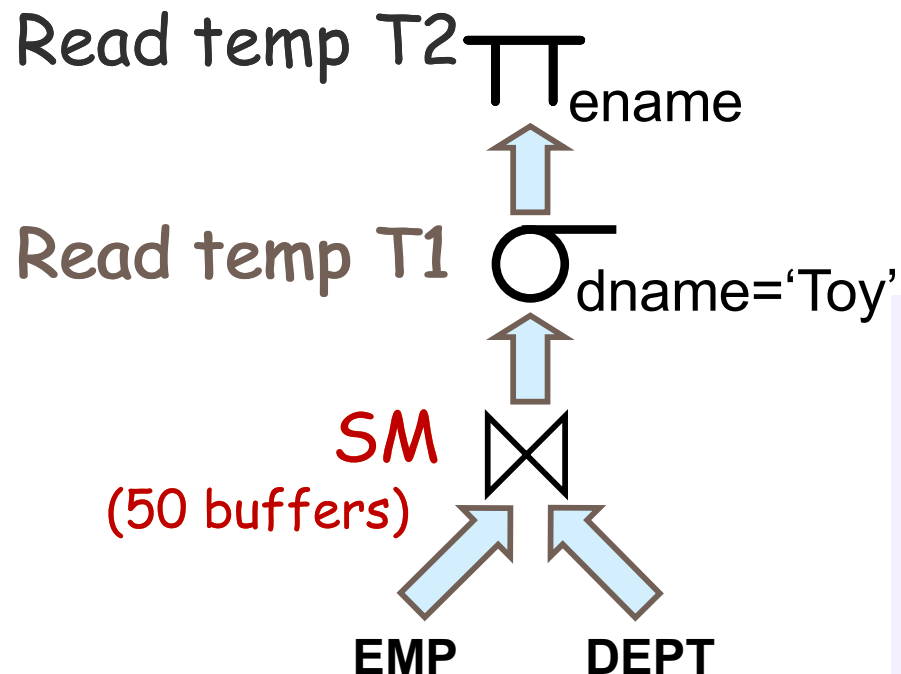
  DEPT (did, dname, floor, mgr)  
 500 departments  
 50 pages



# Example

EMP (ssn, ename, addr, sal, did)  
10,000 employees  
1,000 pages

DEPT (did, dname, floor, mgr)  
500 departments  
50 pages







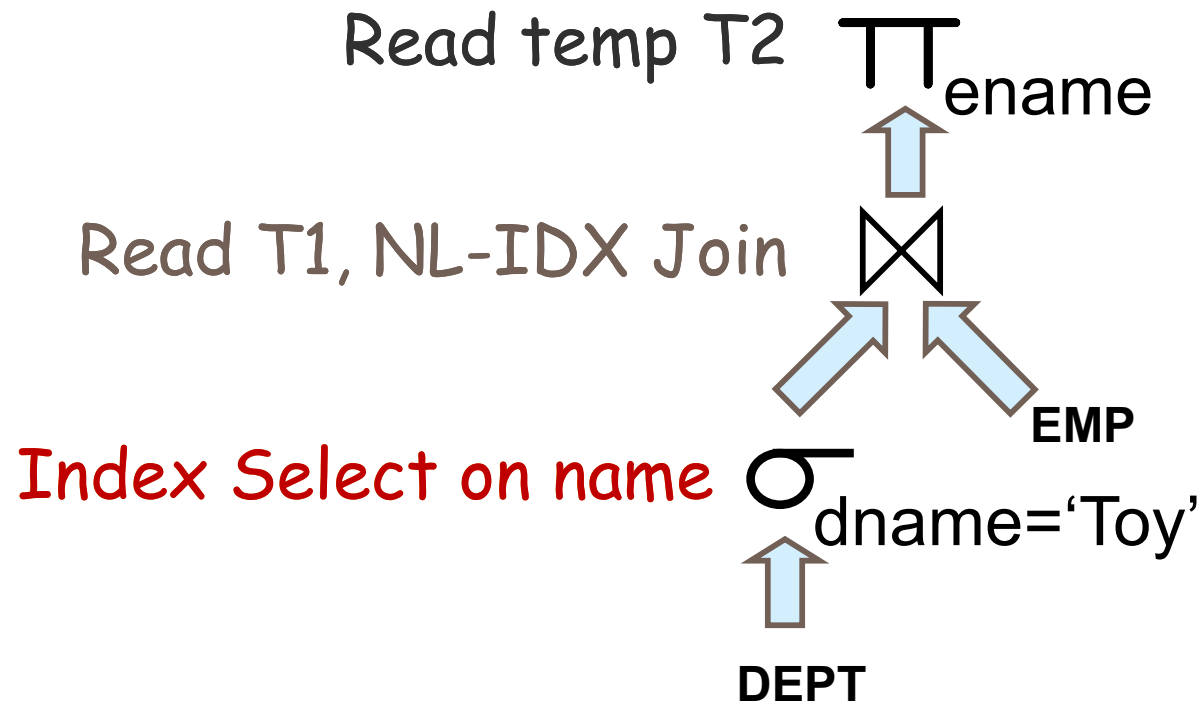
Operator Interface:

- Open
- GetNext
- Close

Supports **both** paradigms

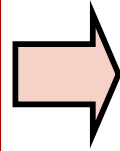
# Example

EMP (ssn, , , addr, sal, )      DEPT (, , dname, floor, mgr)  
10,000 employees      500 departments  
1,000 pages      50 pages



# Query Optimization

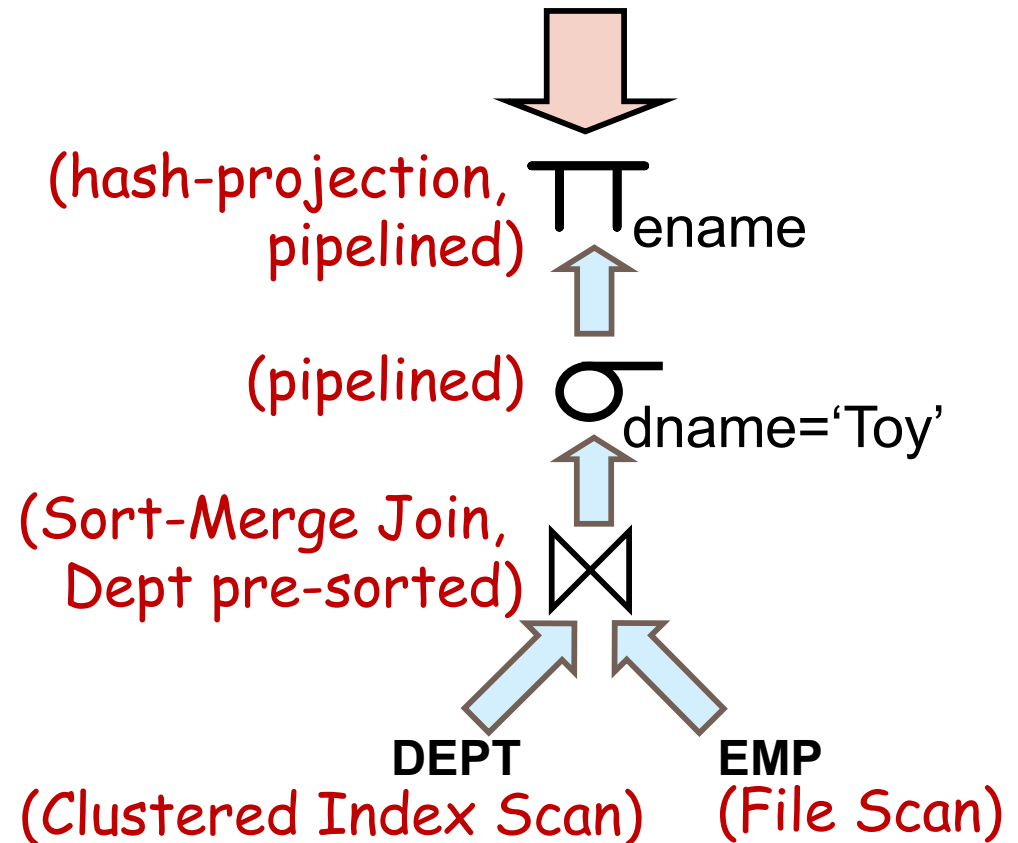
Select distinct ename  
from Emp E, Dept D  
where E.did = D.did  
and D.dname = 'Toy'



$\Pi_{\text{ename}} \sigma_{\text{dname} = \text{'Toy'}} (\text{EMP} \bowtie \text{DEPT})$

1. Identify candidate equivalent trees
2. For each candidate find best annotated version
3. Choose the best overall

Practically: Choose from a **subset** of all possible plans

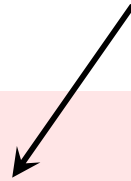


Annotated RA Tree

# Extended RA

HAVING<sub>MAX(SALARY) > 2</sub>( ... )  
GROUP BY<sub>D.did</sub>( ... )

```
SELECT E.did, Max (E.Salary)
FROM Emp E
WHERE addr LIKE '%Palo Alto%'
GROUP BY E.did
HAVING count(*) > 10
```



$\Pi_{\text{did}, \text{Max}(\text{salary})}$   
Having<sub>count(\*) > 10</sub>(Group By<sub>did</sub> ( $\sigma_{\text{addr LIKE '...'}}$  EMP))

Simplification: Only optimize the  $\sigma$ ,  $\Pi$ ,  $\times$

- Project Group By/Having attributes
- Choose from different aggregate algorithms

# Overview of Query Optimization

- *Plan: Annotated RA Tree*
  - Operator interface: Open/getNext/close
  - Pipelined or materialized
- Two main issues:
  - What plans are considered?
    - Algorithm to search plan space for cheapest (estimated) plan.
  - How is the cost of a plan estimated?
- Ideally: Want to find best plan.
- Practically: Avoid worst plans! Look at a subset of all plans

# Cost Estimation

- Estimate *cost* of each operation in plan tree.
  - Depends on input cardinalities.
  - Algorithm cost (see previous lectures)
- Estimate *size of result*
  - Use information about the input relations.
  - For selections and joins, assume independence of predicates.
- We'll discuss the **System R** cost estimation approach.
  - Very inexact, but works ok in practice.
  - More sophisticated techniques known now.

# Pricing Plans: Statistics

- Statistics stored in the catalogs
  - Relation
    - Cardinality
    - Size in pages
  - Index
    - Cardinality (# distinct keys)
    - Size in pages
    - Height
    - Range
- Catalogs update periodically
  - Can be slightly inconsistent
- Commercial systems use histograms
  - More accurate estimates

# Size Estimation and Reduction Factors

```
SELECT attribute list  
FROM relation list  
WHERE term1 AND ... AND termk
```

Question: What is the cardinality of the result set?

- Max # tuples: product of input relation cardinalities
- Each term “filters” out some tuples: *Reduction factor*
- *Result cardinality* = Max # tuples \* product of all RF's.
- **Assumption: terms are independent!**
- Term  $col=value$       RF:  $1/NKeys(I)$ , given index  $I$  on  $col$
- Term  $col1=col2$       RF:  $1/MAX(NKeys(I1), NKeys(I2))$
- Term  $col>value$       RF:  $(High(I)-value)/(High(I)-Low(I))$

# Equivalence

- $\sigma_{p_1} (\sigma_{p_2}(R)) \equiv \sigma_{p_2} (\sigma_{p_1}(R))$  ( $\sigma$  commutativity)
- $\sigma_{p_1 \wedge p_2 \dots \wedge p_n} (R) \equiv \sigma_{p_1}(\sigma_{p_2}(\dots \sigma_{p_n}(R)))$  (cascading  $\sigma$ )
- $\Pi_{a_1}(R) \equiv \Pi_{a_1}(\Pi_{a_2}(\dots \Pi_{a_k}(R)\dots))$ ,  $a_i \subseteq a_{i+1}$  (cascading  $\Pi$ )
- $R \bowtie S \equiv S \bowtie R$  (commutativity)
- $R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T$  (associativity)
- $\sigma_P (R \bowtie S) \equiv (R \bowtie_P S)$  (if  $P$  is a join predicate)
- $\sigma_P (R \bowtie S) \equiv \sigma_{p_1} (\sigma_{p_2}(R) \bowtie_{p_4} \sigma_{p_3}(S))$   $P = p_1 \wedge p_2 \wedge p_3 \wedge p_4$
- $\Pi_{A_1, A_2, \dots, A_n}(\sigma_P (R)) \equiv \Pi_{A_1, A_2, \dots, A_n}(\sigma_P (\Pi_{A_1, \dots, A_n, B_1, \dots, B_M} R))$   
 $B_1 \dots B_M$  attributes in  $P$

# System R Optimizer

- Most widely used currently; works well for  $< 10$  joins
- Cost estimation: Approximate art at best.
  - Catalog statistics
    - cost of operation
    - result size
  - Combination of CPU and I/O costs.
- Plan Space:
  - Only *left-deep plans*
  - Avoid Cartesian products

# Query Blocks: Units of Optimization

- SQL query =>  
collection of *query blocks*
- Optimize one block at a time.
- Treat nested blocks as calls to a subroutine
  - Execute inner block once per outer tuple!
  - In reality more complex optimization
- For each block, consider the following plans:
  - All available access methods, for each relation in FROM clause.
  - All join permutations of left-deep join trees

```
SELECT S.sname
FROM Sailors S
WHERE S.age IN
    (SELECT MAX (S2.age)
     FROM Sailors S2
     GROUP BY S2.rating)
```

*Outer block*    *Nested block*

# Plan Enumeration

- Two main cases:
  - Single-relation plans
  - Multiple-relation plans
- Single relation plan (no joins). Access Plans:
  - file scan
  - index scan(s): Clustered, Non-clustered
    - More than one index may “match” predicates
    - e.g. Clustered index I matching one or more selects:  
Cost:  $(NPages(I) + NPages(R)) * \text{product of RF's of matching selects.}$
  - Choose the one with the least estimated cost.
  - Merge/pipeline selection and projection (and aggregate)
    - RID intersection techniques
    - Index aggregate evaluation

# Example

```
SELECT E.ename  
FROM   Emp E  
WHERE  E.did=8  
AND    E.sal > 40K
```

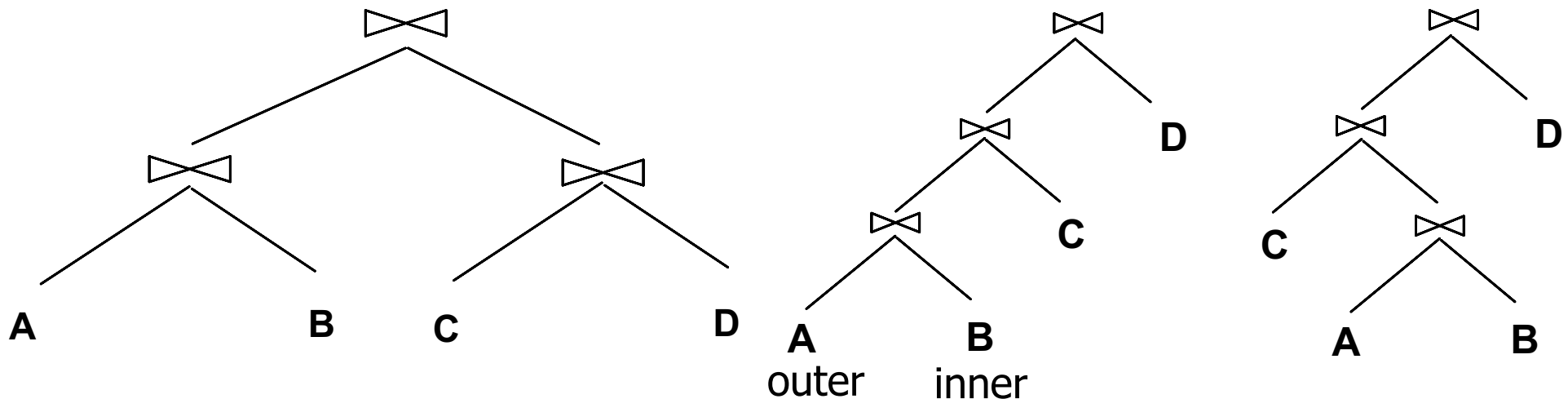
EMP ( ssn,  ename,  addr,  sal,  did)

1,000 data pages, 10K tuples  
100 pages in B+-tree  
# depts: 10  
Salary Range: 10K – 200K

- Index on did:
  - Tuples Retrieved:  $(1/10) * 10,000$
  - Clustered index:  $(1/10) * (100+1,000)$  pages
  - Unclustered index:  $(1/10) * (100+10,000)$  pages
- Index on sal:
  - Clustered index:  $(200-40)/(200-10) * (100+1,000)$  pages
  - Unclustered index: ...
- File scan: 1,000 pages

# Queries Over Multiple Relations

- System R: Only consider left-deep join trees
  - Used to restrict the search space
  - Left-deep plans can be *fully pipelined plans*.
    - Intermediate results not written to temporary files.
    - Not all left-deep trees are fully pipelined (e.g., SM join).



Linear Tree: at least 1 child in every join node is a base relation

# Enumeration of Left-Deep Plans

- Decide:
  - Join order
  - Join method for each join
- Enumerated using N passes (if N relations joined):
  - Pass 1: Find best 1-relation plan for each relation.
  - Pass 2: Find best way to join result of each 1-relation plan (as outer) to another relation. (*All 2-relation plans.*)
  - Pass N: Find best way to join result of a (N-1)-relation plan (as outer) to the N'th relation. (*All N-relation plans.*)
- For each subset of relations, retain only:
  - Cheapest plan overall, plus
  - Cheapest plan for each *interesting order* of the tuples.

$\Pi_{\text{ename}} \sigma_{\text{dname} = \text{'Toy'}} (\text{EMP} \bowtie \text{DEPT})$

EMP (**ssn**, ename, addr, sal, **did**)

DEPT (**did**, **dname**, floor, mgr)

Pass 1: EMP: E1: S(EMP), E2: I (EMP.did)

*Cost:*

DEPT: D1: S(DEPT), D2: **I.(DEPT.did)**, D3: **I(DEPT.dname)**

*Cost:*

Pass 2: Consider EMP  $\bowtie$  DEPT and DEPT  $\bowtie$  EMP

EMP  $\bowtie$  DEPT, Alternatives:

1. E1  $\bowtie$  D2: Algorithms ...
2. E1  $\bowtie$  D3: Algorithms ...
3. E2  $\bowtie$  D2: Algorithms **SM**, NL, BNL, NL-IDX, Hash
4. E2  $\bowtie$  D3: Algorithms

Similarly consider DEPT  $\bowtie$  EMP

**Pick cheapest 2-relation plan. Done (with join optimization)**

**Next Consider GROUP BY (if present) ...**

# Enumeration of Plans (Contd.)

- ORDER BY, GROUP BY handled as a final step,
- Only “join” relations if there is a connecting join condition i.e., avoid Cartesian products if possible.
- This approach is still exponential in the # of tables.

# Summary

- Query optimization critical to the DBMS performance
  - Helps understand performance impact of database design
- Two parts to optimizing a query:
  - Enumerate alternative plans. Typically, only consider left-deep plans
  - Estimate cost of each plan: size of result and cost of algorithm
    - *Key issues*: Statistics, indexes, operator implementations.
- Single-relation queries: Pick cheapest access plan + interesting order
- Multiple-relation queries:
  - All single-relation plans are first enumerated. Selections/projections considered as early as possible.
  - For each 1-relation plan, consider all ways of joining another relation (as inner)
  - Keep adding 1-relation plan until done
  - At each level, retain cheapest plan, and best plan for each interesting order