



Query Optimization

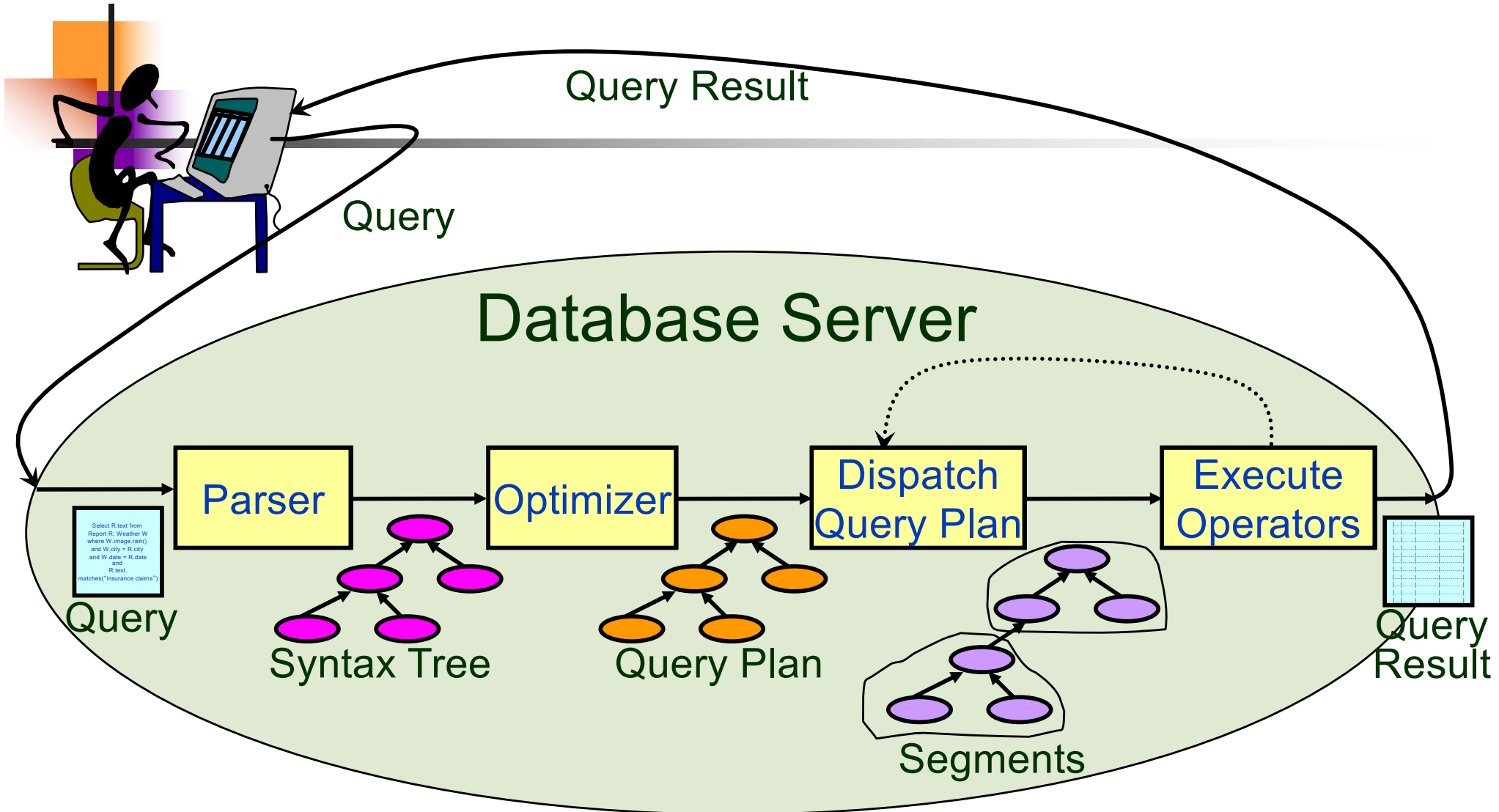
Chapter 15



Query Optimization

- Given a SQL query, how do we evaluate it efficiently?
- *Query Optimizer* – Important component of a DBMS
 - Convert SQL query *blocks* to extended relational algebra expressions
 - Enumerate alternative evaluation plans
 - Choose a plan based on estimated cost

Query Execution Life-Cycle



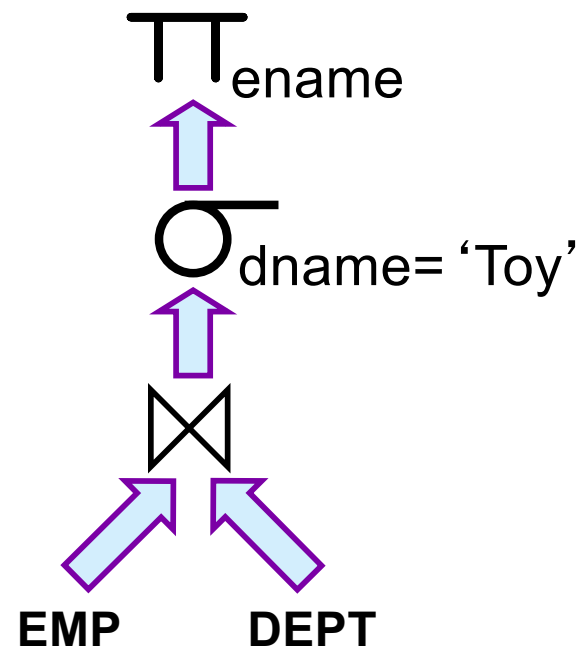
Query Evaluation Plan

- Extended relational algebra tree, including algorithms for each operator

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

DEPT (did, dname, floor, mgr)

SELECT E.ename
FROM Emp E, Dept D
WHERE D.dname = 'Toy'
AND D.did = E.did



Query Optimizer selects the evaluation plan



Query Evaluation Plan

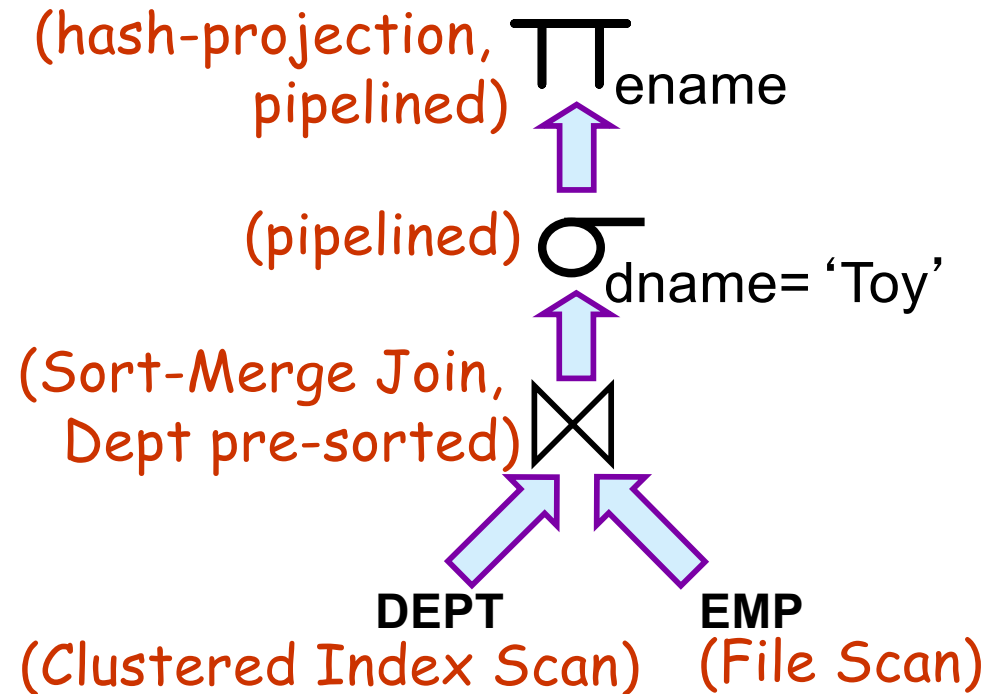
- Annotated, extended RA tree
- Intermediate Results (multiple ops):
 - **Pipelined**: Tuples resulting from one operator fed directly into the next
 - **Materialized**: Create a temporary table to store intermediate results

Example

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

DEPT (did, dname, floor, mgr)

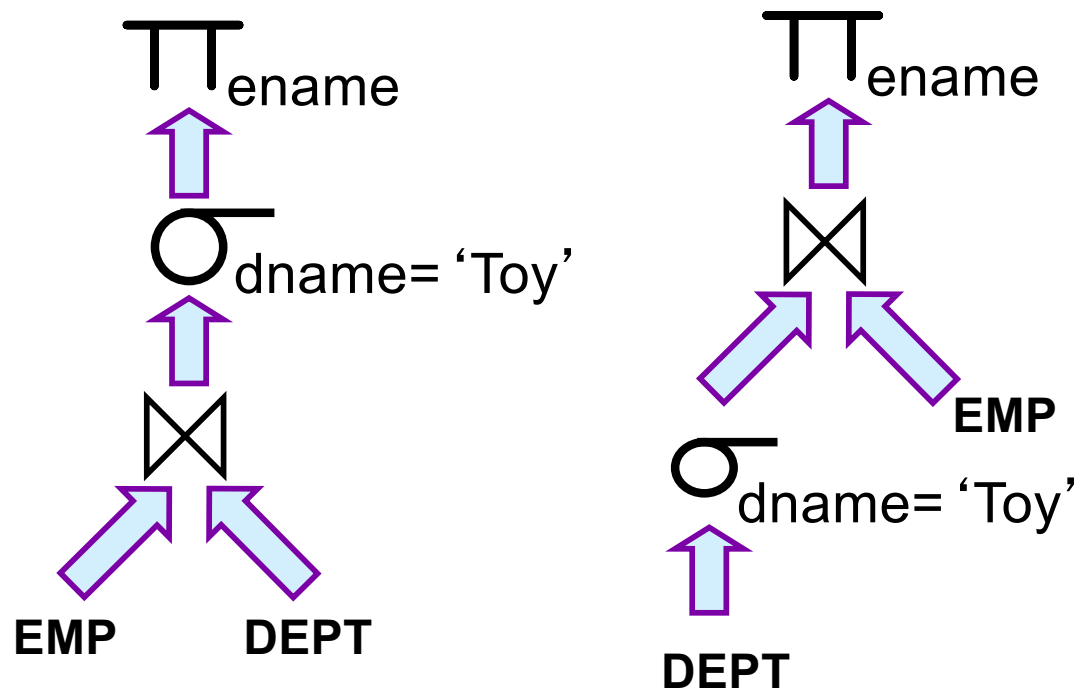
```
SELECT DISTINCT E.ename
FROM Emp E, Dept D
WHERE D.dname = 'Toy'
AND D.did = E.did
```



Annotated RA Tree

Alternative Plans

- Can be many extended RA trees that produce the same result!



Also, different algorithms for each operator

Extended RA

HAVING_{MAX(SALARY) > 2(...)}
GROUP BY_{D.did (...)}

```
Select E.did, Max (E.Salary)
From   Emp E
Where  addr = 'Palo Alto'
Group By E.did
Having count(*) > 10
```

$\Pi_{\text{did}, \text{Max(salary)}}$
Having_{count(*) > 10}(Group By_{did} ($\sigma_{\text{addr} = \text{'Palo Alto'}}$ EMP))

Simplification: Only optimize the σ , Π , χ

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



RA Equivalence - Selections

- $\sigma_{P_1} (\sigma_{P_2}(R)) \equiv \sigma_{P_2} (\sigma_{P_1}(R))$ (σ 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 σ)

Selection operation is commutative and multiple selections on the same relation can be combined into a single selection.



RA Equivalence – Projections

- $\Pi_{a_1}(R) \equiv \Pi_{a_1}(\Pi_{a_2}(\dots \Pi_{a_k}(R)\dots))$, $a_i \subseteq a_{i+1}$ (cascading Π)
- In the above, each a_i is a set of columns.
- For example: Let's say R has columns c_1, c_2, c_3, c_4 .
- $\Pi_{c_1, c_3}(R) \equiv (\Pi_{c_1, c_3}(\Pi_{c_1, c_2, c_3}(R)))$

Basically, we are eliminating one column at a time.

The above is useful when optimizing expressions that involve both projections and joins.



RA Equivalence: cross-products & joins

$$R \bowtie S \equiv S \bowtie R \text{ (commutativity)}$$

$$R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T \text{ (associativity)}$$

Note: Join without any common attributes is same as cross-product

$$R \times S \equiv S \times R \text{ (commutativity)}$$

$$R \times (S \times T) \equiv (R \times S) \times T \text{ (associativity)}$$

- Thus, joins and cross products can be performed in any order
- Same columns will result. Column order not important semantically. (SQL can reorder the columns at final presentation time to the user, if column order is relevant for UI purposes.)



RA Equivalence – Multiple Ops

- $\Pi_A(\sigma_c(R)) \equiv \sigma_c(\Pi_A(R))$ (if selection only involves attributes in the set A)
- $\sigma_p(R \times S) \equiv (R \bowtie_p S)$
- $\sigma_p(R \times S) \equiv \sigma_p(R) \times S$ (if p is only on R)
- $\sigma_p(R \bowtie S) \equiv \sigma_p(R) \bowtie S$ (if p is only on R)

Key ideas: When possible, consider doing

- Joins rather than cross-products.
- Selections before joins.
- Projections before selections.



RA Equivalence – Multiple Ops

- $\sigma_P (R \bowtie S) \equiv \sigma_{P_4} (\sigma_{P_1}(R) \bowtie_{p_3} \sigma_{P_2}(S))$
- Note: $P = p_1 \wedge p_2 \wedge p_3 \wedge p_4$

Idea:

Replace P by a cascade of conditions p1, p2, p3, and p4 such that:

P1: conditions only on R

P2: conditions only on S

P3: join conditions with equality on R and S

P4: other conditions involving both R and S columns



RA Equivalence – Multiple Ops

- $\Pi_p (R \times S) \equiv \Pi_{p_1}(R) \times \Pi_{p_2}(S)$

Columns p in the cross-product consist of columns p_1 from R and columns p_2 from S .

- $\Pi_p (R \bowtie S) \equiv \Pi_{p_1}(R) \bowtie \Pi_{p_2}(S)$

Note: The above is only possible if all the join attributes appear in both p_1 and p_2 .



RA Equivalence – More rules

- $\Pi_{A1,A2,\dots,A_n}(\sigma_P(R)) \equiv \Pi_{A1,A2,\dots,A_n}(\sigma_P(\Pi_{A1,\dots,A_n, B1,\dots,B_M}R))$
B1 ... BM attributes in P

- $\Pi_A(R \bowtie_c S) \equiv \Pi_A((\Pi_{A1}(R) \bowtie_c \Pi_{A2}(S)))$

A1 is the set of attributes of R in either A or in c.

A2 is the set of attributes of S in either A or in c.



RA Equivalence

- Additional equivalences possible when union, intersection, set difference, etc. are considered.
- We do not discuss those further.



Query Optimization – Main Issues

- Which (equivalent) plans do we consider?
- How do we estimate the cost of a plan?
- **Ideally:** Want to find best plan
- **Practically:** Avoid worst plans! Look at a subset of all plans
- Typical optimizations
 - Re-ordering joins
 - “Pushing” selections and projections



Cost Based Optimization

- Most widely used currently; works well for < 10 joins
- **Cost estimation:** Approximate at 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

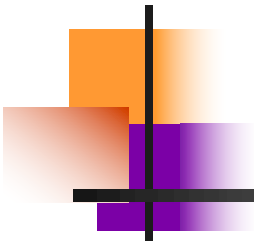


System Catalogs

- For each relation:
 - name, file name, file structure (e.g., Heap file)
 - attribute name and type, for each attribute
 - index name, for each index
 - integrity constraints
- For each index:
 - structure (e.g., B+ tree) and search key fields
- For each view:
 - view name and definition
- Plus statistics, authorization, buffer pool size, etc.

Catalogs are themselves stored as relations

Example Attribute_Cat Catalog



attrName	relName	type	position
attrName	Attribute_Cat	string	1
relName	Attribute_Cat	string	2
type	Attribute_Cat	string	3
position	Attribute_Cat	integer	4
sid	Students	string	1
name	Students	string	2
login	Students	string	3
age	Students	integer	4
gpa	Students	real	5
fid	Faculty	string	1
fname	Faculty	string	2
sal	Faculty	real	3



Cost Estimation

- Estimate *cost* of each operation in plan tree
 - Depends on input cardinalities
 - Algorithm cost (see previous lecture)
- 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

- Example statistics stored in the catalogs (e.g., pg_stats and pg_class in Postgres)
 - Relation
 - Cardinality (# rows)
 - 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

Example

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

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

Compute the cost of different plans:

- Index on did:
 - Tuples Retrieved: ?
 - Clustered index: ?
 - Unclustered index: ?
- Index on sal:
 - Clustered index: ?
 - Unclustered index: ...
- File scan: ?

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



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: $(High(I)-value)/(High(I)-Low(I))$

(If no index, replace Nkeys with # of distinct values for that attribute.)



Plan Enumeration

- Two main cases:
 - Single-relation plans
 - Multiple-relation plans
- Single-relation plan (no joins). Access Plans:
 - file scan
 - index scan(s): Clustered, Unclustered
 - 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 aggregation)
 - Consider using RID intersection techniques
 - Consider using Index aggregate evaluation

Example

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

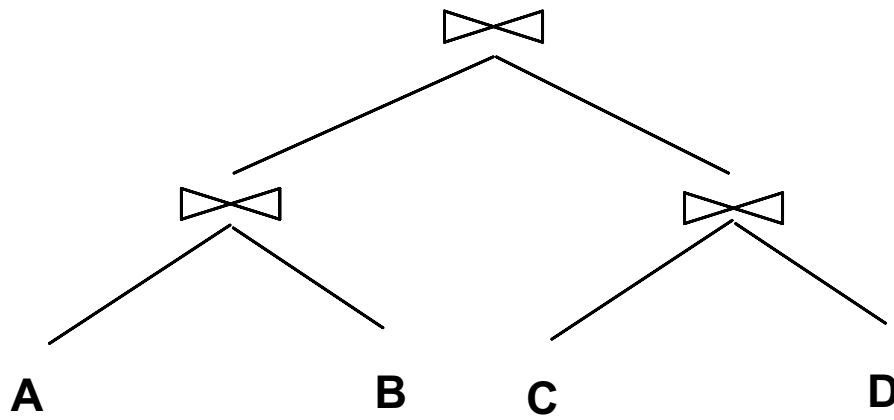
EMP (ssn, ename, addr, sal, did)

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

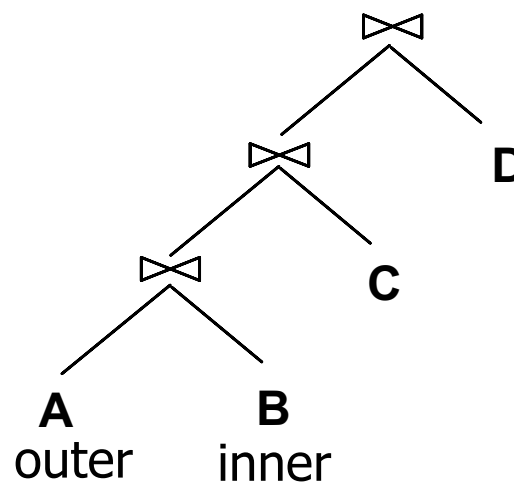
- Index on did:
 - Tuples Retrieved: $(1/10) * 10,000$
 - Clustered index: $1 + (1/10) * (1,000)$ pages
 - Unclustered index: $(1/10) * (100+10,000)$ pages
- Index on sal:
 - Clustered index: $(200-40)/(200-10) * (100+1,000)$ pages
 - Unclustered index: figure this out.
- File scan: 1,000 pages
- Intersecting Rids? Could use both indexes, intersect rids, and then fetch the pages. (figure this out.)

Queries Over Multiple Relations

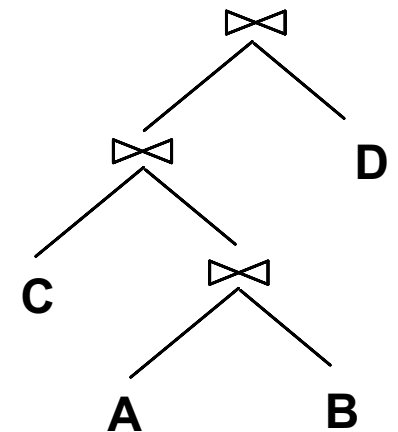
- System R: Only consider *left-deep* join trees
 - Used to restrict the search space
 - Left-deep plans can be *fully pipelined*.
 - Intermediate results not written to temporary files.
 - Not all left-deep trees are fully pipelined (e.g., SM join when the input relations are not sorted).



Not left-deep join tree



Left-deep join tree



Not left-deep join tree



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 (apply selections and projections first and consider using indexing)
- For each relation, retain only:
 - Cheapest plan overall (e.g., File scan), plus
 - Cheapest plans that produce *ordered* tuples (e.g., using a B+-tree index). Order may be useful for a later sort-merge join, even though the plan may be more expensive at this point.



Enumeration of Left-Deep Plans

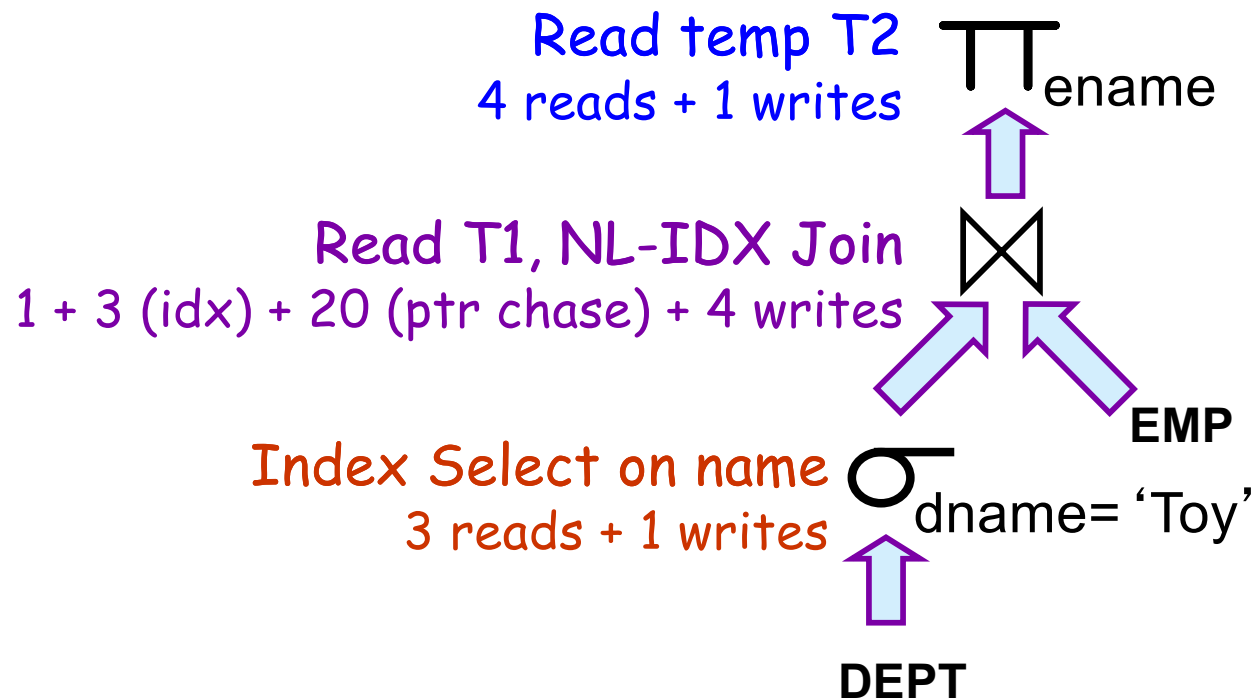
- **Pass 2:** Find best way to join result of each 1-relation plan (as outer) to another relation. (*All 2-relation plans.*)
- **Pass N:** Find best way to join result of a (N-1)-relation plan (as outer) to the N'th relation. (*All N-relation plans.*)
- For each subset of relations, retain only:
 - Cheapest plan overall, plus
 - Cheapest plan for each *interesting order* of the tuples.
- Also, apply selections first, where possible
- Apply projections first as well, where possible
- Assume pipelining of results to reduce

Example

EMP (ssn, ename, addr, sal, did) DEPT (did, dname, floor, mgr)

10,000 employees 500 departments

1,000 pages 50 pages



Total: 37 I/Os

Example

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

EMP (ssn, ename, addr, sal, did) DEPT (did, dname, floor, mgr)

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

Cost: 1000 1000+100

KEEP E1 and E2!

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

Cost: 50 50+5

3+5 **KEEP D2 and D3**

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) ...

Example

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

EMP (ssn, ename, addr, sal, did) DEPT (did, dname, floor, mgr)

10,000 employees 500 departments

1,000 pages 50 pages

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

Total: 2M I/Os






4 reads, 1 write $\rho (R, \Pi_{\text{ename}} T3)$

2,000 + 4 writes $\rho (T3, \sigma_{\text{dname} = \text{'Toy'}} T2)$
(10K/500 = 20 emps per dept)

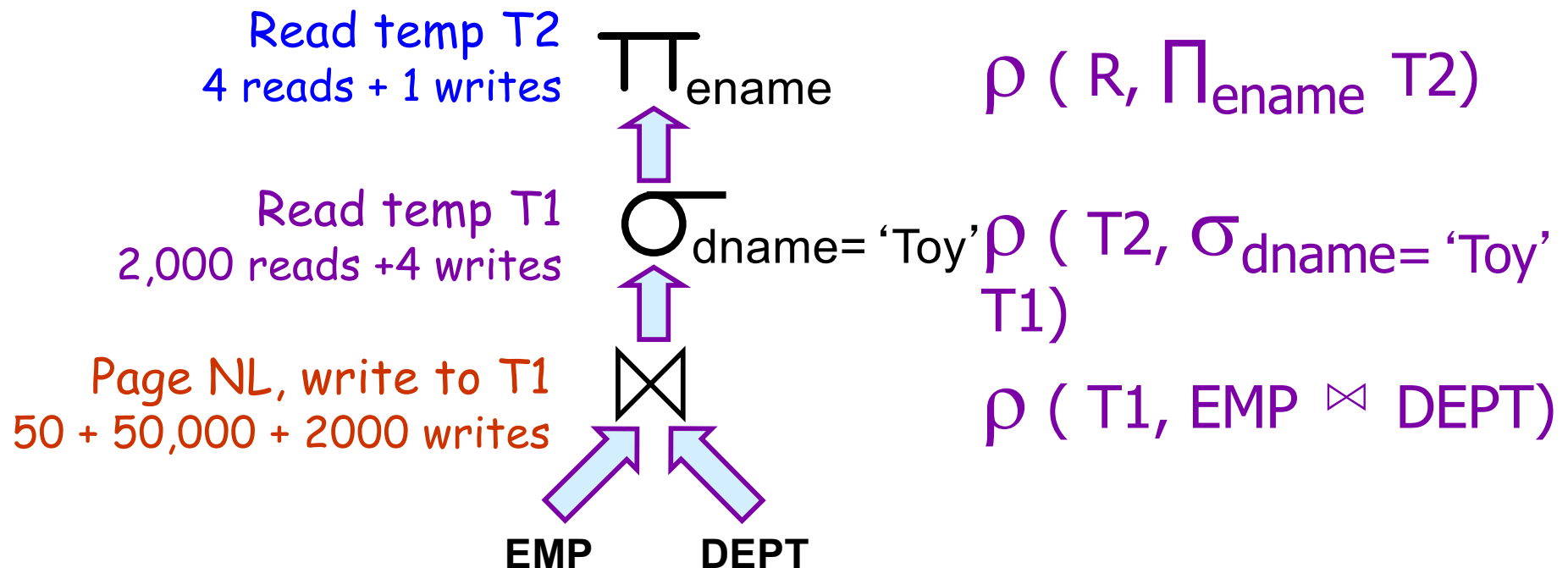
1,000,000 + 2,000 writes $\rho (T2, \sigma_{\text{EMP.did} = \text{DEPT.did}} T1)$
(FK join, 10K tuples in T2)

50 + 50,000 + 1,000,000 writes $\rho (T1, \text{EMP} \times \text{DEPT})$
(5 tuples per page in T1)






Example

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

Total: 54K I/Os



Example

EMP (ssn, ename, addr, sal, did) DEPT (did, dname, floor, mgr)

10,000 employees 500 departments
 1,000 pages 50 pages


Read temp T2
 4 reads + 1 writes

 ename

Read temp T1
 2,000 reads + 4 writes

 dname = 'Toy'

SM
 (50 buffers)
 3 (M+N) = 3150 + 2000 writes


 EMP DEPT

With **Materialization**
 Total: 7,159 I/Os

With **Pipelining**
 Total: 3,151 I/Os

Operator Interface:

- Open
- GetNext
- Close

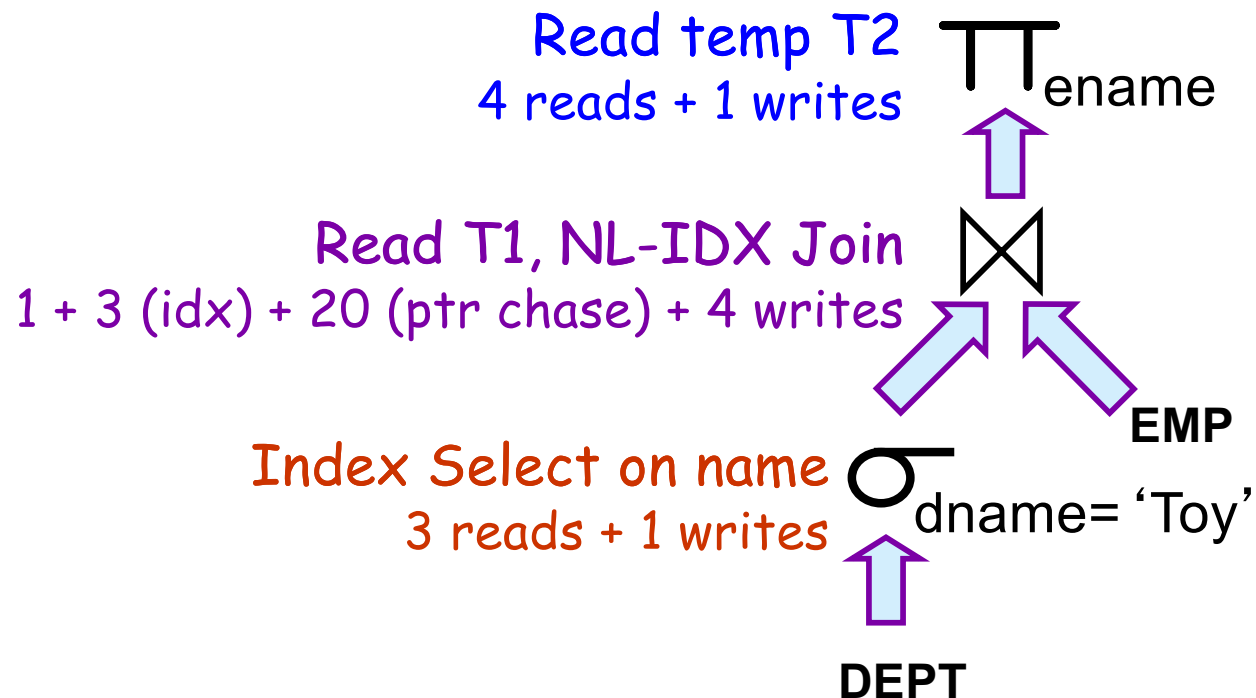
Supports **both** paradigms

Example

EMP (ssn, ename, addr, sal, did) DEPT (did, dname, floor, mgr)

10,000 employees 500 departments

1,000 pages 50 pages



Total: 37 I/Os



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.

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*



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



Announcements

Optional Exercises: 12.1 (all parts), 15.1, 15.5, 15.7, 15.9