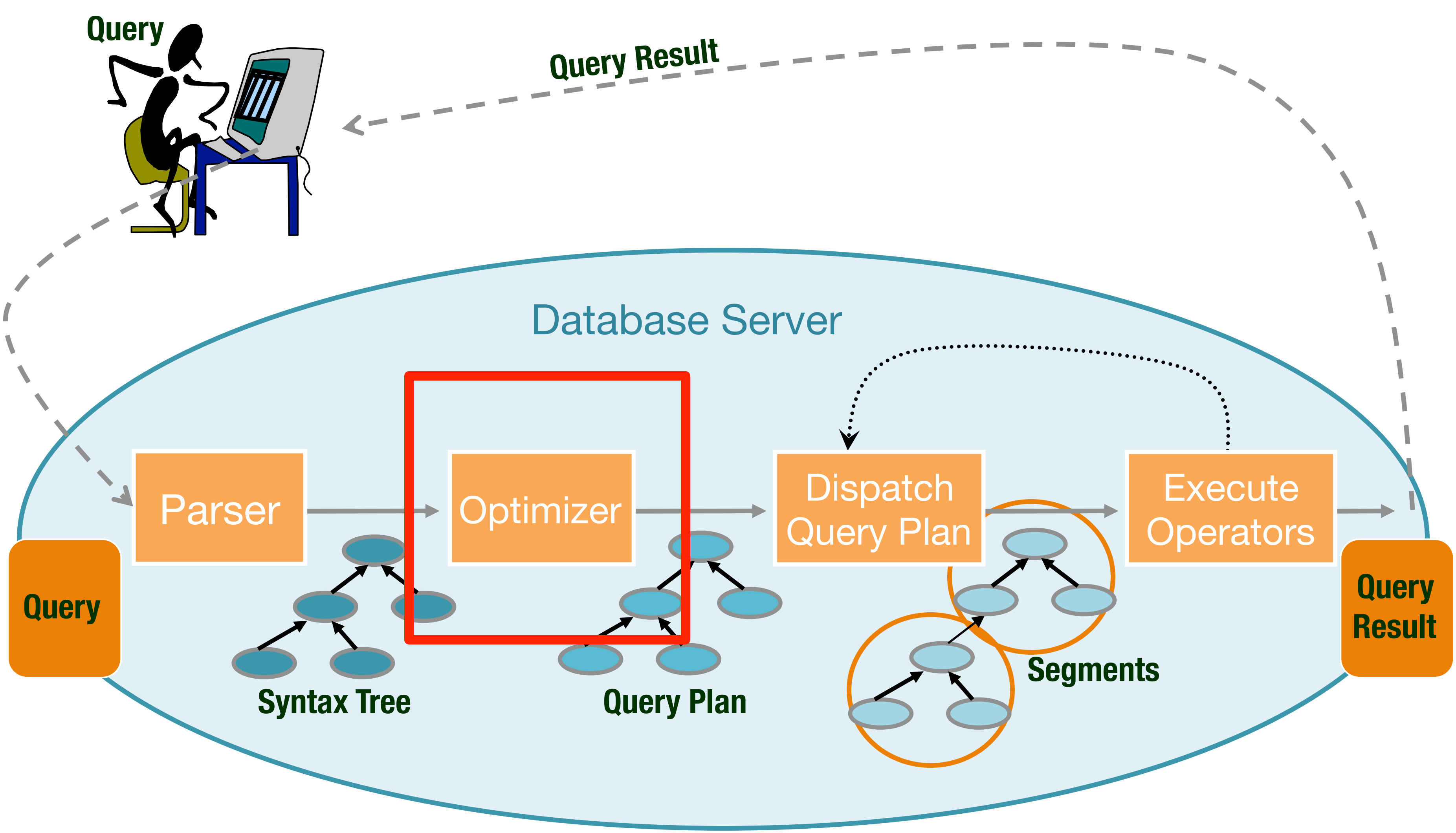
Query Optimization

Chapter 15

Query Execution Life-Cycle



Query Optimization

Query optimizer selects an evauation plan with the lease cost in two steps:

1. **Plan Enumeration**

Today

- Different query plans
- Different implementations (i.e., evaluation algorithms) for each operator

Previous lectures

2. **Cost Estimation**

- Cost of each operator
- Overall cost of the plan

Annotated RA Expressions

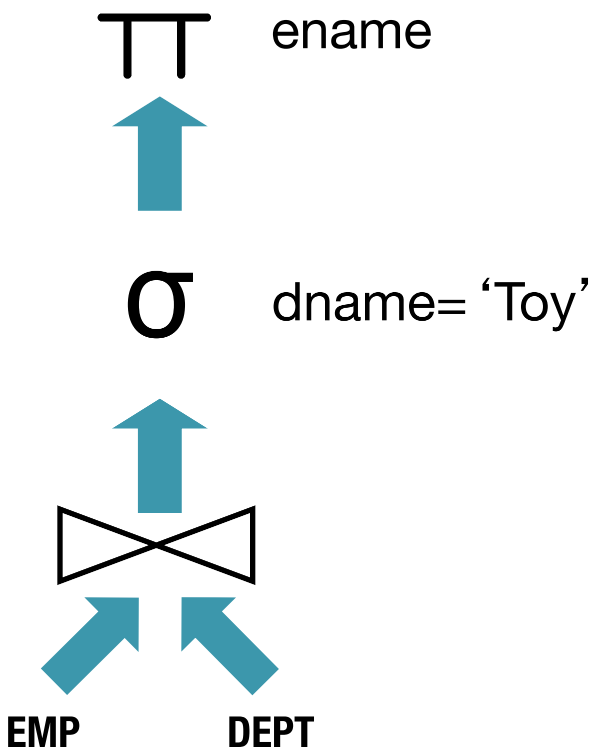
- Which algorithm to use for each operator
- Where Intermediate Results are:
  - Pipelined: Tuples resulting from one operator fed directly into the next
  - Materialized: Create a temporary table to store intermediate results

Example: RA Tree

```
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
```

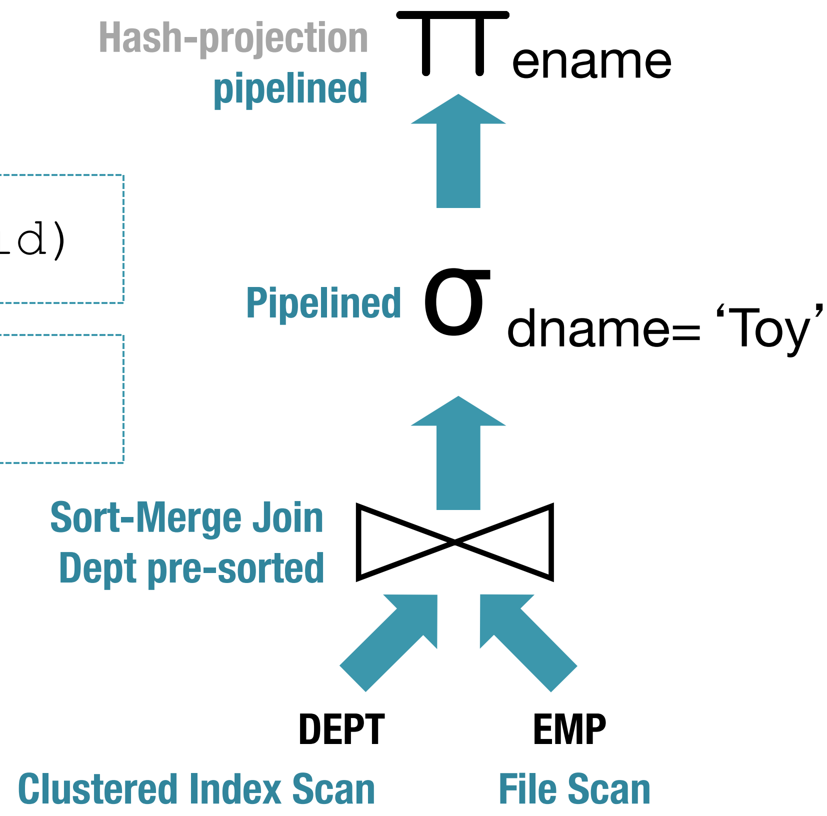


Example: Annotated RA Tree

```
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
```



Annotated RA Tree

Extended RA

HAVING $_{\text{MAX(SALARY) > 2}}$(…)

GROUP BY $_{\text{D.did}}$ (…)

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

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

Simplification: Only optimize the σ, Π, X

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

RA Equivalence - Selections

$\sigma_{P1}(\sigma_{P2}(R)) \equiv \sigma_{P2}(\sigma_{P1}(R))$ (σ commutativity)

$\sigma_{P1 \wedge P2 \ldots \wedge Pn}(R) \equiv \sigma_{P1}(\sigma_{P2}(\ldots \sigma_{Pn}(R)))$ (cascading σ)

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

RA Equivalence – Projections

$\prod_{a1}(R) \equiv \prod_{a1}(\prod_{a2}(\ldots\prod_{ak}(R)\ldots)),\ a_i \subseteq a_{i+1}$ (cascading $\prod$)

- In the above, each ai is a set of columns.
- For example: Let's say R has columns c1, c2, c3, c4

$\prod_{c1,c3}(R) \equiv (\prod_{c1,c3}(\prod_{c1,c2,c3}(R))$

- 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

$R \bowtie S \equiv S \bowtie R$ (commutativity)

$R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T$ (associativity)

$R \times S \equiv S \times R$ (commutativity)

$R \times (S \times T) \equiv (R \times S) \times T$ (associativity)

- => joins and cross products can be performed in any order
- Same rows 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

$$\sigma_{P}\ (R \times S) \equiv \sigma_{P4}\ (\sigma_{P1}(R) \bowtie_{p3} \sigma_{P2}(S))$$

Note: $P = p1 \wedge p2 \wedge p3 \wedge p4$

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

Example

$\sigma_P\ (R \times S) \equiv \sigma_{P4}\ (\sigma_{P1}(R)\ \bowtie_{p3}\ \sigma_{P2}(S))$

Note: P=p1 ∧ p2 ∧ p3 ∧ p4

$\sigma_{major=CSE\ \wedge\ R.umid=S.umid\ \wedge\ grade>C}\ (R \times S) \equiv$
$\sigma_{true}\ (\sigma_{major=cse}(R)\ \bowtie_{umid}\ \sigma_{grade>c}(S))$

umid	sname	major
23	Alice	CSE
71	Bob	ECE
11	Mary	CSE
13	John	CSE

umid	course	semes	grade
13	484	W17	A+
23	484	W17	A+
71	484	W17	A+
23	482	W17	C
71	482	W17	D
11	482	W17	D-

RA Equivalence – Multiple Ops

$$\Pi_P (R \times S) \equiv \Pi_{P1}(R) \times \Pi_{P2}(S)$$

Columns p in the cross-product consist of columns p1 from R and columns p2 from S

Why is this useful?

$$\Pi_P (R \bowtie_c S) \equiv \Pi_P (\Pi_{P1}(R) \bowtie_c \Pi_{P2}(S))$$

p1 are attrs of R that appear in p or c

p2 are attrs of S that appear in p or c

Example

$$\Pi_P (R \bowtie_c S) \equiv \Pi_P (\Pi_{P1}(R) \bowtie_c \Pi_{P2}(S))$$

$$\Pi_{sname,\ course} (R \bowtie_{umid} S)$$
$$\equiv \Pi_{sname,\ course} (\Pi_{sname,umid}(R) \bowtie_{umid} \Pi_{course,umid}(S))$$

umid	sname	major
23	Alice	CSE
71	Bob	ECE
11	Mary	CSE
13	John	CSE

umid	course	semes	grade
13	484	W17	A+
23	484	W17	A+
71	484	W17	A+
23	482	W17	C
71	482	W17	D
11	482	W17	D-

RA Equivalence – More Rules

$$\prod_{A1,A2,\ldots An}(\sigma_P(R)) \equiv \prod_{A1,A2,\ldots An}(\sigma_P(\prod_{A1,\ldots An,\ B1,\ldots\ BM}R))$$

B1 … BM attributes in P

$$\prod_{umid}(\sigma_{major=ECE}(R)) \equiv \prod_{umid}(\sigma_{major=ECE}(\prod_{umid,major}R))$$

umid	sname	major
23	Alice	CSE
71	Bob	ECE
11	Mary	CSE
13	John	CSE

RA Equivalence

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

Types of Optimization

- Rule-Based
  - If we know that plan A and plan B are equivalent, and that plan A is likely to be cheaper, then we should replace plan B with plan A.
  - E.g. Push selections in.
  - E.g. Push projections in.
- Cost-Based
  - If we can get (good) estimates of the cost of equivalent plans A and B, we can prefer the one that has lower cost.
  - Can use this even if there is no universal best choice.
  - E.g. choosing between join methods.
  - Major innovation in database systems

Query Optimization – Main Issues

Q1: Which (equivalent) plans do we consider?

Q2: How do we estimate the cost of a plan?

- Ideal Goal: Find the best plan
- Pragmatic Goal: Avoid worst plans!
- Typical optimizations
  - Re-ordering joins
  - “Pushing” selections and projections
    - e.g., avoid cartesian products

Cost Based Optimization

- Most popular currently; works well for < 10 joins
- Cost estimation: Approximate at best
  - Use statistics from systems catalogs
  - Combination of CPU and I/O costs
    - But this class focuses on I/O costs only
    - We also use **System R** approach
      - very inexact but ok in practice (better techniques are known now)

Estimating the Cost of a Plan

1. Estimate *cost* of each operation in plan tree
    - Depends on input cardinalities (# of rows)
    - Algorithm cost (see previous lectures)
2. Estimate *size of result*
    - Use information about the input relations
    - For selections and joins, assume independence of predicates

Why care about result sizes?

Collecting & Maintaining Statistics

- Statistics stored in the catalogs (pg_stats & 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

Estimating Output Size

```
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, Nkeys = # distinct values

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)) * product of RF's of matching selects
  - Choose the one with the least estimated cost.
  - Merge/pipeline selection and projection (and aggregation)
    - RID intersection techniques
    - Index aggregate evaluation (e.g. min or max age)

Quiz: Cost of different plans

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

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

Estimate the cost of different access methods:

- Index on did:
  - Clustered index: ?
  - Unclustered index: ?
- Index on sal:
  - Clustered index:?
  - Unclustered index: …
- File scan: ?

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

Quiz: Cost of different plans

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

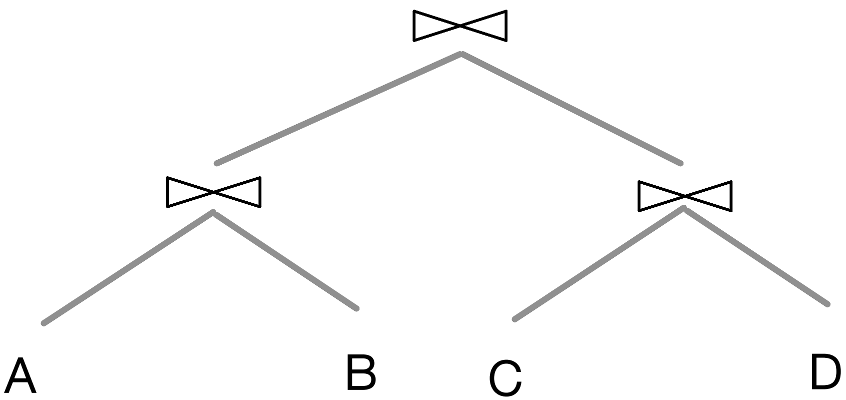
- Index on did:
  - Tuples Retrieved: (1/10) * 10,000
  - Clustered index: (1/10) * (100+1,000) pages
  - Unclustered index: (1/10) * (100+10,000) pages
- Index on sal:
  - Clustered index: (200-40)/(200-10) * (100+1,000) pages
  - Unclustered index: (200-40)/(200-10) * (100+10,000) pages
- File scan: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

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

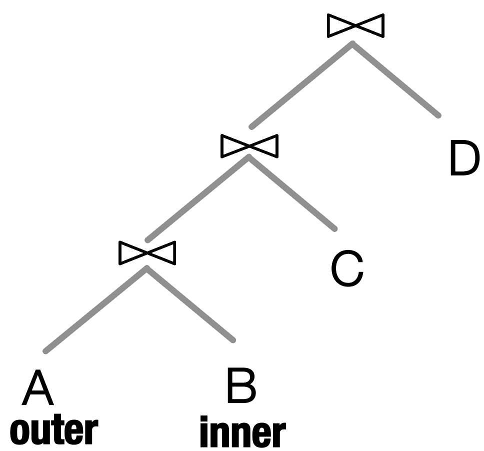
1,000 data pages, 10K tuples
100 pages in B+-tree
depts: 10
Salary Range: 10K – 200K

Multiple-Relation Plans

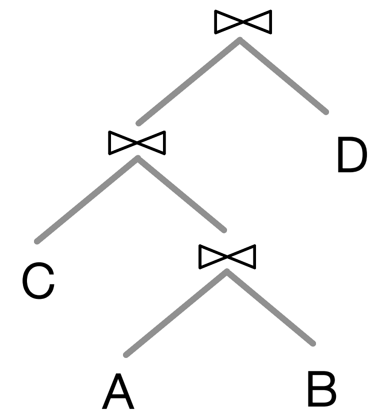
- System R: Only consider left-deep join trees
  - Used to restrict the search space
  - Left-deep plans can be fully pipelined (usually)
    - Intermediate results not written to temporary files
    - Not all left-deep trees are fully pipelined (e.g., SM join)



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 avoid additional I/O

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

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

Optional Exercises

12.1 (all parts), 15.1, 15.5, 15.7, 15.9