Lectures 9 -10: Query optimization
Announcements

• HW3 (SimpleDB) is due next Friday!

• Reading assignment was due today
Query Optimization Motivation

SQL query

Parse & Rewrite Query

Select Logical Plan

Select Physical Plan

Query Execution

Disk

Declarative query

Recall physical and logical data independence

Logical plan

Physical plan

Query optimization
What We Already Know

• There exists many logical plans...

• ... and for each, there exist many physical plans

• Optimizer chooses the logical/physical plan with the smallest estimated cost
Discussion of the Paper

• Query parsing/authorization
• Query rewriting:
  – Is salary < 75k and salary > 100k implausible?
  – What is semantic optimization?
• Query optimizer
  – Will discuss in detail...
  – What is query re-optimization?
    Predictable performance (IBM) v.s. self-tuning (Microsoft)
  – What is the “halloween problem”?
• Query execution
  – What are BP-tuples v.s. M-tuples? What is the pin-count?
• Access methods: will discuss
Query Optimization

Three major components:

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 each physical operator if we knew the $T(R)$ and $B(R)$ for each of its arguments

- Goal: estimate $T(R)$ for each intermediate result $R$. $B(R)$ can be derived from $T(R)$
Statistics on Base Data

• Collected information for each database 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
\[ T(\Pi(R)) = T(R) \]

**Selection:** the size decreases by *selectivity factor* \( \theta \)
\[ T(\sigma_{\text{pred}}(R)) = T(R) \times \theta_{\text{pred}} \]
Selectivity Factors

- **A = c**  
  \[/* \sigma_{A=c}(R) */\]  
  - Selectivity = \(1/V(R,A)\)

- **A < c**  
  \[/* \sigma_{A<c}(R)*/\]  
  - Selectivity = \((c - \min(R, A))/(\max(R,A) - \min(R,A))\)

- **c1 < A < c2**  
  \[/* \sigma_{c1<A<c2}(R)*/\]  
  - Selectivity = \((c2 – c1)/(\max(R,A) - \min(R,A))\)

- **Multiple predicates: assume independence**
Estimating Result Sizes

Join \( R \bowtie_{R.A=S.B} S \)

- Take product of cardinalities of relations \( R \) and \( S \)

- Apply this selectivity factor:
  \[
  1/ ( \max( V(R,A), V(S,B) ) )
  \]

- Why? Will explain next...
Assumptions

• **Containment of values**: if $V(R,A) \leq V(S,B)$, 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 $V(S, C)$)
  
  – This is only needed higher up in the plan
Selectivity of $R \bowtie_{A=B} S$

Assume $V(R,A) \leq V(S,B)$

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

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

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

• Extensive example next...
Logical Query Plan 1

\[
\pi_{\text{sname}}
\]

\[
\sigma_{\text{pno}=2 \land \text{scity}='Seattle' \land \text{sstate}='WA'}
\]

\[
\text{SELECT sname}
\FROM \text{Supplier } x, \text{Supply } y
\WHERE x.\text{sid} = y.\text{sid}
\AND y.\text{pno} = 2
\AND x.\text{scity} = 'Seattle'
\AND x.\text{sstate} = 'WA'
\]

\[
T(\text{Supplier}) = 1000
\]

\[
B(\text{Supplier}) = 100
\]

\[
V(\text{Supplier, scity}) = 20
\]

\[
V(\text{Supplier, state}) = 10
\]

\[
M=11
\]
Logical Query Plan 1

\[ \sigma_{\text{pno}=2 \land \text{scity}=\text{'Seattle'} \land \text{sstate}='\text{WA}'}(\text{Supplier}) \]

\[ \pi_{\text{sname}}(\sigma_{\text{pno}=2 \land \text{scity}=\text{'Seattle'} \land \text{sstate}='\text{WA}'}(\text{Supplier})) \]

\[ \text{SELECT sname FROM Supplier} \times, \text{Supply} y \text{WHERE} x.\text{sid} = y.\text{sid} \text{and} y.\text{pno} = 2 \text{and} x.\text{scity} = \text{'Seattle'} \text{and} x.\text{sstate} = \text{'WA'} \]

Estimated (why?)

T = 10000

M = 11

\[ \text{T(Supplier)} = 1000 \]
\[ \text{B(Supplier)} = 100 \]
\[ \text{V(Supplier, scity)} = 20 \]
\[ \text{V(Supplier, state)} = 10 \]

\[ \text{T(Supply)} = 10000 \]
\[ \text{B(Supply)} = 100 \]
\[ \text{V(Supply, pno)} = 2500 \]
Logical Query Plan 1

\[ \Pi_{sname} \sigma_{pno=2 \land scity='Seattle' \land sstate='WA'} \]

\[ T < 1 \]

\[ T = 10000 \]

\[ \text{Estimated (why?)} \]

\[ \text{SELECT } sname \]
\[ \text{FROM } \text{Supplier } x, \text{Supply } y \]
\[ \text{WHERE } x.sid = y.sid \]
\[ \text{and } y.pno = 2 \]
\[ \text{and } x.scity = 'Seattle' \]
\[ \text{and } x.sstate = 'WA' \]

\[ T(Supply) = 10000 \]
\[ B(Supply) = 100 \]
\[ V(Supply, pno) = 2500 \]

\[ T(Supplier) = 1000 \]
\[ B(Supplier) = 100 \]
\[ V(Supplier, scity) = 20 \]
\[ V(Supplier, state) = 10 \]

\[ M=11 \]
Logical Query Plan 2

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

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, sstate) = 10

M=11
Logical Query Plan 2

\[
\text{SELECT sname}
\text{FROM Supplier x, Supply y}
\text{WHERE x.sid = y.sid}
\text{and y.pno = 2}
\text{and x.scity = 'Seattle'}
\text{and x.sstate = 'WA'}
\]
Logical Query Plan 2

\[
\begin{align*}
\text{SELECT} & \quad \text{sname} \\
\text{FROM} & \quad \text{Supplier} \ x, \ \text{Supply} \ y \\
\text{WHERE} & \quad x.\text{sid} = y.\text{sid} \quad \text{and} \quad y.\text{pno} = 2 \\
& \quad \text{and} \quad x.\text{scity} = \text{’Seattle’} \\
& \quad \text{and} \quad x.\text{sstate} = \text{’WA’}
\end{align*}
\]

\(T(\text{Supplier}) = 1000\)
\(B(\text{Supplier}) = 100\)
\(V(\text{Supplier}, \text{scity}) = 20\)
\(V(\text{Supplier}, \text{sstate}) = 10\)

\(M = 11\)
Logical Query Plan 2

\[
\begin{align*}
T(Supplier) &= 10000 \\
B(Supplier) &= 100 \\
V(Supplier, pno) &= 2500
\end{align*}
\]

\[
\begin{align*}
T(Supply) &= 10000 \\
B(Supply) &= 100 \\
V(Supply, pno) &= 2500
\end{align*}
\]

\[
\begin{align*}
M &= 11
\end{align*}
\]

\[
\begin{align*}
\text{SELECT} & \quad \text{sname} \\
\text{FROM} & \quad \text{Supplier} \ x, \ \text{Supply} \ y \\
\text{WHERE} & \quad x.\text{sid} = y.\text{sid} \\
& \quad \text{and} \ y.\text{pno} = 2 \\
& \quad \text{and} \ x.\text{scity} = \text{‘Seattle’} \\
& \quad \text{and} \ x.\text{sstate} = \text{‘WA’}
\end{align*}
\]

Very wrong! Why?

\[
\begin{align*}
\Pi_{\text{sname}} & \quad T = 4 \\
\sigma_{\text{pno}=2} & \quad T = 4 \\
\sigma_{\text{scity}=\text{‘Seattle’} \land \text{sstate}=\text{‘WA’}} & \quad T = 5
\end{align*}
\]

\[
\begin{align*}
\text{Supplier}(\text{sid, sname, scity, sstate}) \\
\text{Supply}(\text{sid, pno, quantity})
\end{align*}
\]
Logical Query Plan 2

\[
\begin{align*}
\text{SELECT } & \text{sname} \\
\text{FROM } & \text{Supplier x, Supply y} \\
\text{WHERE } & \text{x.sid = y.sid} \\
& \text{y.pno = 2} \\
& \text{x.scity = 'Seattle'} \\
& \text{x.sstate = 'WA'}
\end{align*}
\]

M = 11

\[
\begin{align*}
\text{T(Supplier)} & = 10000 \\
\text{B(Supplier)} & = 100 \\
\text{V(Supplier, scity)} & = 20 \\
\text{V(Supplier, sstate)} & = 10
\end{align*}
\]

\[
\begin{align*}
\text{T(Supply)} & = 10000 \\
\text{B(Supply)} & = 100 \\
\text{V(Supply, pno)} & = 2500
\end{align*}
\]
Physical Plan 1

\[
\begin{align*}
\Pi_{\text{sname}} & \quad \text{Scan} \\
\sigma_{\text{pno}=2 \land \text{scity}='Seattle' \land \text{sstate}='WA'} & \quad \text{Scan} \\
\text{sid} = \text{sid} & \quad \text{Block nested loop join} \\
\end{align*}
\]

\[
\begin{align*}
\text{T} & = 10000 \\
\text{B} & = 100 \\
\text{V(\text{Supplier, pno})} & = 2500 \\
\text{T(\text{Supplier})} & = 1000 \\
\text{B(\text{Supplier})} & = 100 \\
\text{V(\text{Supplier, scity})} & = 20 \\
\text{V(\text{Supplier, state})} & = 10 \\
\end{align*}
\]

Total cost: \(\frac{100}{10} \times 100 = 1000\)
Physical Plan 1

\[ \Pi_{\text{sname}} \sigma_{\text{pno}=2 \land \text{scity}='Seattle' \land \text{sstate}='WA'}(\text{Supplier})\]

\[ \text{Total cost: } 100 + 100 \times 100/10 = 1100 \]

\[ \text{T(Supplier)} = 1000 \]
\[ \text{B(Supplier)} = 100 \]
\[ \text{V(Supplier, pno)} = 2500 \]

\[ \text{M=11} \]

\[ \text{Supply} \]
\[ \text{Scan} \]
\[ \text{T(Supply)} = 10000 \]
\[ \text{B(Supply)} = 100 \]

\[ \text{Supplier} \]
\[ \text{Scan} \]
\[ \text{sid = sid} \]
\[ \text{Block nested loop join} \]
Physical Plan 2

\[ \Pi_{\text{sname}} (\sigma_{\text{sstate} = 'WA'} \pi_{\text{sname}} (\text{Supplier} \cup \text{Supply} \leftarrow \text{sid} = \text{sid})) \]

\[ \text{Cost of Supply(pno)} = 4 \]
\[ \text{Cost of Supplier(scity)} = 50 \]
\[ \text{Total cost: 54} \]

\[ \sigma_{\text{sstate} = 'WA'} \]
\[ \sigma_{\text{scity} = 'Seattle'} \]

\[ T(\text{Supply}) = 10000 \]
\[ B(\text{Supply}) = 100 \]
\[ V(\text{Supply, pno}) = 2500 \]
\[ T(\text{Supplier}) = 1000 \]
\[ B(\text{Supplier}) = 100 \]
\[ V(\text{Supplier, scity}) = 20 \]
\[ V(\text{Supplier, state}) = 10 \]

\[ M = 11 \]
Physical Plan 2

\[ \pi_{\text{sname}}(\sigma_{\text{pno} = 2}(\text{Supply})) \]

\[ \sigma_{\text{sstate} = 'WA'}(\text{Supplier}) \]

\[ \sigma_{\text{scity} = 'Seattle'}(\text{Supplier}) \]

Cost of Supply(pno) = 4
Cost of Supplier(scity) = 50
Total cost: 54

Unclustered index lookup
Supplier(scity)

Unclustered index lookup
Supply(pno)

Main memory join

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

\( M = 11 \)
Physical Plan 2

\[ \text{T(Supplier)} = 1000 \]
\[ \text{B(Supplier)} = 100 \]
\[ \text{V(Supplier, scity)} = 20 \]
\[ \text{V(Supplier, state)} = 10 \]
\[ \text{T(Supply)} = 10000 \]
\[ \text{B(Supply)} = 100 \]
\[ \text{V(Supply, pno)} = 2500 \]

\( \text{M} = 11 \)

Cost of Supply(pno) = 4
Cost of Supplier(scity) = 50
Total cost: 54
Physical Plan 3

\[ \sigma_{\text{city}='Seattle' \land \text{sstate}='WA'} \]

\[ \Pi_{\text{name}} \]

\[ \sigma_{\text{pno}=2} \]

Unclustered index lookup
Supply(pno)

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

Cost of Supply(pno) = 4
Cost of Index join = 4
Total cost: 8

Clustered Index join

\[ \sigma_{\text{city}='Seattle' \land \text{sstate}='WA'} \]

T = 4
sid = sid

M = 11
Physical Plan 3

$\text{Supplier}(\text{sid, sname, scity, sstate})$
$\text{Supply}(\text{sid, pno, quantity})$

$\Pi_{\text{sname}} \sigma_{\text{scity}='Seattle' \land \text{sstate}='WA'}$

$\sigma_{\text{pno}=2}$

$\text{Supply}$

$\text{Supplier}$

Cost of $\text{Supply(pno)} = 4$
Cost of Index join = 4
Total cost:

Unclustered index lookup $\text{Supply(pno)}$

Clustered Index join

$T(\text{Supply}) = 10000$
$B(\text{Supply}) = 100$
$V(\text{Supply, pno}) = 2500$

$T(\text{Supplier}) = 1000$
$B(\text{Supplier}) = 100$
$V(\text{Supplier, scity}) = 20$
$V(\text{Supplier, state}) = 10$

$M=11$
Physical Plan 3

\[ \pi_{sname} \sigma_{scity='Seattle' \land sstate='WA'} \]

Cost of Supply(pno) = 4
Cost of Index join = 4
Total cost: 8

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

M=11
R×S in Postgres

Courtesy of Walter Cai

Join algorithm choice varying over base relation size

- Nested loop
- Hash join
- Merge join
Simplifications

• We considered only IO cost; in general we need IO+CPU

• We assumed that all index pages were in memory: sometimes we need to add the cost of fetching index pages from disk
Histograms

- Statistics on data maintained by the RDBMS
- Makes size estimation much more accurate (hence, cost estimations are more accurate)
Employee(ssn, name, age)

\[ T(\text{Employee}) = 25000, \quad V(\text{Employee}, \text{age}) = 50 \]
\[ \min(\text{age}) = 19, \quad \max(\text{age}) = 68 \]

\[ \sigma_{\text{age}=48}(\text{Employee}) = ? \quad \sigma_{\text{age}>28 \text{ and age}<35}(\text{Employee}) = ? \]
Employee(ssn, name, age)

\( T(\text{Employee}) = 25000, \ V(\text{Employee}, \text{age}) = 50 \)
\( \min(\text{age}) = 19, \ \max(\text{age}) = 68 \)

\( \sigma_{\text{age}=48}(\text{Employee}) = ? \quad \sigma_{\text{age}>28 \text{ and age}<35}(\text{Employee}) = ? \)

\( \text{Estimate} = 25000 / 50 = 500 \quad \text{Estimate} = 25000 \times 6 / 50 = 3000 \)
Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Employee, age) = 50
min(age) = 19, max(age) = 68

\( \sigma_{\text{age}=48}(\text{Employee}) = ? \) \( \sigma_{\text{age}>28 \text{ and age}<35}(\text{Employee}) = ? \)

<table>
<thead>
<tr>
<th>Age:</th>
<th>0..20</th>
<th>20..29</th>
<th>30-39</th>
<th>40-49</th>
<th>50-59</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>200</td>
<td>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
</tbody>
</table>
Histories

Employee(ssn, name, age)

\( T(\text{Employee}) = 25000, \ V(\text{Employee}, \text{age}) = 50 \)
\( \min(\text{age}) = 19, \ \max(\text{age}) = 68 \)

\( \sigma_{\text{age}=48}(\text{Employee}) = ? \)
\( \sigma_{\text{age}>28 \text{ and age}<35}(\text{Employee}) = ? \)

<table>
<thead>
<tr>
<th>Age</th>
<th>0..20</th>
<th>20..29</th>
<th>30-39</th>
<th>40-49</th>
<th>50-59</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>200</td>
<td>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
</tbody>
</table>

Estimate = 1200  Estimate = 1*80 + 5*500 = 2580
Types of Histograms

• How should we determine the bucket boundaries in a histogram?
Types of Histograms

- How should we determine the bucket boundaries in a histogram?
  - Eq-Width
  - Eq-Depth
  - Compressed
  - V-Optimal histograms
**Employee**: (ssn, name, age)

### Histograms

#### Eq-width:

<table>
<thead>
<tr>
<th>Age:</th>
<th>0..20</th>
<th>20..29</th>
<th>30-39</th>
<th>40-49</th>
<th>50-59</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>200</td>
<td>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
</tbody>
</table>

#### Eq-depth:

<table>
<thead>
<tr>
<th>Age:</th>
<th>0..20</th>
<th>20..29</th>
<th>30-39</th>
<th>40-49</th>
<th>50-59</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>1800</td>
<td>2000</td>
<td>2100</td>
<td>2200</td>
<td>1900</td>
<td>1800</td>
</tr>
</tbody>
</table>

**Compressed**: store separately highly frequent values: (48,1900)
V-Optimal Histograms

- Defines bucket boundaries in an optimal way, to minimize the error over all point queries
- Computed rather expensively, using dynamic programming
- Modern databases systems use V-optimal histograms or some variations
Discussion

• Small number of buckets
  – Hundreds, or thousands, but not more
  – WHY ?

• *Not* updated during database update, but recomputed periodically
  – WHY ?

• Multidimensional histograms rarely used
  – WHY ?
Query Optimization

**Three major components:**

1. Cardinality and cost estimation

2. Search space
   - Access path selection
   - Rewrite rules

3. Plan enumeration algorithms
Access Path

**Access path**: a way to retrieve tuples from a table

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

Usually the access path implements a selection $\sigma_P(R)$, where the predicate $P$ is called *search argument* SARG (see paper)
Access Path Selection

Supplier(sid,sname,scity,sstate)
Selection condition: sid > 300 ∧ scity='Seattle'
Indexes: clustered B+-tree on sid; B+-tree on scity

V(Supplier,scity) = 20
Max(Supplier, sid) = 1000, Min(Supplier,sid) =1
B(Supplier) = 100, T(Supplier) = 1000

Which access path should we use?
Access Path Selection

\[
\text{Supplier}(\text{sid}, \text{sname}, \text{scity}, \text{sstate})
\]
Selection condition: \( \text{sid} > 300 \land \text{scity} = 'Seattle' \)
Indexes: clustered B+-tree on \text{sid}; B+-tree on \text{scity}

\[
V(\text{Supplier}, \text{scity}) = 20 \\
\text{Max}(\text{Supplier, sid}) = 1000, \ \text{Min}(\text{Supplier,sid}) = 1 \\
\text{B(Supplier)} = 100, \ \text{T(Supplier)} = 1000
\]

Which access path should we use?

1. Sequential scan: cost = 100
Access Path Selection

**Supplier(sid,sname,scity,sstate)**
Selection condition: $sid > 300 \land scity='Seattle'$
Indexes: clustered B+-tree on $sid$; B+-tree on $scity$

$V(Supplier,scity) = 20$
$Max(Supplier, sid) = 1000$, $Min(Supplier,sid) = 1$
$B(Supplier) = 100$, $T(Supplier) = 1000$

Which access path should we use?

1. Sequential scan: cost = 100
2. Index scan on $sid$: cost = $\frac{7}{10} \times 100 = 70$
Access Path Selection

Supplier(sid, sname, scity, sstate)
Selection condition: sid > 300 \ And \ scity='Seattle'
Indexes: clustered B+-tree on sid; B+-tree on scity

\[ V(\text{Supplier}, \text{scity}) = 20 \]
\[ \text{Max}(\text{Supplier}, \text{sid}) = 1000, \ \text{Min}(\text{Supplier}, \text{sid}) = 1 \]
\[ B(\text{Supplier}) = 100, \ T(\text{Supplier}) = 1000 \]

Which access path should we use?

1. Sequential scan: cost = 100
2. Index scan on sid: cost = 7/10 * 100 = 70
3. Index scan on scity: cost = 1000/20 = 50
Rewrite Rules

- The optimizer’s search space is defined by the set of rewrite rules that it implements
- More rewrite rules means that more plans are being explored
Relational Algebra Laws

• **Selections**
  – Commutative: $\sigma_{c_1}(\sigma_{c_2}(R))$ same as $\sigma_{c_2}(\sigma_{c_1}(R))$
  – Cascading: $\sigma_{c_1 \land c_2}(R)$ same as $\sigma_{c_2}(\sigma_{c_1}(R))$

• **Projections**
  – Cascading

• **Joins**
  – Commutative: $R \Join S$ same as $S \Join R$
  – Associative: $R \Join (S \Join T)$ same as $(R \Join S) \Join T$
Selections and Joins

\[ \sigma_{A=v}(R(A, B) \bowtie_{B=C} S(C, D)) = \]

\( R(A, B), \ S(C, D) \)
Selections and Joins

\( \sigma_{A=v}(R(A,B) \bowtie_{B=C} S(C,D)) = (\sigma_{A=v}(R(A,B))) \bowtie_{B=C} S(C,D) \)

The simplest optimizers use only this rule
Called *heuristic-based optimizer*
In general: *cost-based optimizer*
Group-by and Join

\[ \gamma_{A, \text{sum}(D)}(R(A,B) \Join_{B=C} S(C,D)) \]
Group-by and Join

\[ \gamma_{A, \text{sum}(D)}(R(A,B) \Join_{B=C} S(C,D)) = \gamma_{A, \text{sum}(D)}(R(A,B) \Join_{B=C} (\gamma_{C, \text{sum}(D)} S(C,D))) \]

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
Left-Deep Plans and Bushy Plans

Left-deep plan

Bushy plan
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

\[
\begin{align*}
R(A,B) & \quad \text{SELECT} \quad * \\
S(B,C) & \quad \text{FROM} \quad R, S, T \\
T(C,D) & \quad \text{WHERE} \quad R.B=S.B \text{ and } S.C=T.C \text{ and } R.A<40
\end{align*}
\]

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

Why is this best for rewrite rules?
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 (1979)

- 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
System R Enumeration Algorithm

- Idea: use dynamic programming
- For each subset of \( \{R_1, \ldots, R_n\} \), compute the best plan for that subset
- In increasing order of set cardinality:
  - Step 1: for \( \{R_1\}, \{R_2\}, \ldots, \{R_n\} \)
  - Step 2: for \( \{R_1, R_2\}, \{R_1, R_3\}, \ldots, \{R_{n-1}, R_n\} \)
  - ...
  - Step n: for \( \{R_1, \ldots, R_n\} \)
- It is a bottom-up strategy
- A subset of \( \{R_1, \ldots, R_n\} \) is also called a subquery
Dynamic Programming Algo.

• For each subquery $Q \subseteq \{R_1, \ldots, R_n\}$ 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 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)