Announcements

• Project meetings this Friday

• HW3 is posted, due next Friday
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
Today

• Discuss Query Optimization

• In parallel, discuss the paper
  How Good Are Query Optimizers, Really? VLDB’2015
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
Query Optimization

Three major components:

1. Cardinality and cost estimation
2. Search space
3. Plan enumeration algorithms
Cost Estimation

Goal: compute cost of an entire physical plan

• We know how to compute the cost given B, T:
  – E.g. index join COST = B(R)+T(R)B(S)/V(S,a)

New Goal: estimate T(R) for each intermediate R
  “Cardinality Estimation”
Cardinality Estimation

Problem: given statistics on base tables and a query, estimate size of the answer

Very difficult, because:
• Need to do it very fast
• Need to use very little memory
Statistics on Base Data

Statistics on base tables

- Number of tuples (cardinality) \( T(R) \)
- Number of physical pages \( B(R) \)
- Indexes, number of keys in the index \( V(R,a) \)
- Histogram on single attribute (1d)
- Histogram on two attributes (2d)

Computed periodically, often using sampling
Assumptions

• Uniformity

• Independence

• Containment of values

• Preservation of values
Size Estimation

**Projection**: output size same as input size
\[ T(\Pi(R)) = T(R) \]

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

*Uniformity assumption*
Selectivity Factors

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

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

Multiple predicates: **independence assumption**

- **A = c and B = d**  
  
  \[
  \sigma_{A=c \text{ and } B=d}(R) \]
  
  - Selectivity  = 1/V(R,A) * 1/V(R,B)
Estimating Result Sizes

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

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

- Apply this selectivity factor:
  \[
  \frac{1}{\text{MAX}( V(R,A), V(S,B))}
  \]

- 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, $V(R \Join_{A=B} S, C) = V(R, C)$ (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 $T(R \bowtie_{A=B} S) = T(R) \cdot T(S) / V(S,B)$

In general:

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

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

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

\[M=11\]
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 1

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

\[ \pi_{sname} \]

Estimated (why?)

T = 10000

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

\[ \pi_{sname} \]

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

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

M=11
Logical Query Plan 1

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

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

\[ T < 1 \]

\[ T = 10000 \]

Estimated (why?)

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

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

\[ M=11 \]
Logical Query Plan 2

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

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

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

\[M=11 \]
Logical Query Plan 2

\[
\sigma_{\text{scity}=\text{Seattle} \land \text{sstate}=\text{WA}} \left( \pi_{\text{sname}} \left( \sigma_{\text{pno}=2} \left( \sigma_{\text{sid}=\text{sid}} \left( \text{Supplier} \times \text{Supply} \right) \right) \right) \right)
\]

M = 11

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

\[
\begin{align*}
T(\text{Supply}) &= 10000 \\
B(\text{Supply}) &= 100 \\
V(\text{Supply, pno}) &= 2500
\end{align*}
\]
Logical Query Plan 2

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

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

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

\[ \text{Supply} \]

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

\[ \text{Very wrong! Why?} \]

\[ \text{M}=11 \]
Logical Query Plan 2

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

\[ \text{M} = 11 \]

\[ \text{T(Supplier)} = 1000 \ \\
\text{B(Supplier)} = 100 \ \\
\text{V(Supplier, scity)} = 20 \ \\
\text{V(Supplier, state)} = 10 \ \\
\text{T(Supply)} = 10000 \ \\
\text{B(Supply)} = 100 \ \\
\text{V(Supply, pno)} = 2500 \]
Logical Query Plan 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'
```

Very wrong! Why?

Different estimate 😞

Supply

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

Supplier

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

M=11
Physical Plan 1

\[ \pi_{sname} \]

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

\[ T = 10000 \]

Block nested loop join

Scan

Supply

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

Scan

Supplier

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

Total cost:

\[ \frac{100}{10} \times 100 = 1000 \]

M=11
Physical Plan 1

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

\[T = 10000\]

\[T < 1\]

Total cost: 100 + 100 * 100/10 = 1100

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

\[M = 11\]

\[
\begin{align*}
\text{T(Supply)} &= 10000 \\
\text{B(Supply)} &= 100 \\
\text{V(Supply, pno)} &= 2500
\end{align*}
\]
Physical Plan 2

\[ \Pi_{\text{sname}} \]

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

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

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

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

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

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

\[ \text{M = 11} \]
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 \( \text{Supply}(\text{pno}) \) = 4
Cost of \( \text{Supplier}(\text{scity}) \) = 50
Total cost: 54

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

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

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

Unclustered index lookup \( \text{Supplier}(\text{scity}) \)

Main memory join

\( M=11 \)
Physical Plan 2

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

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

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

```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

Unclustered index lookup
Supply(pno)

Unclustered index lookup
Supplier(scity)

Main memory join
Physical Plan 3

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

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

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

\[ \text{Supply} \]

\[ \text{Supplier} \]

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

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

Cost of \( \text{Supply(pno)} \) = 4
Cost of Index join = 4
Total cost: 8
Physical Plan 3

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

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

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

\[ \text{Supply} \]

\[ \text{Supplier} \]

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

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

\text{M=11}

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

\text{M=11}
Physical Plan 3

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

Cost of \text{Supply(pno)} = 4
Cost of Index join = 4
Total cost: 8

M=11
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)
Histograms

Employee(ssn, name, age)

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

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

Employee(ssn, name, age)

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

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

Estimate = \( \frac{25000}{50} = 500 \)

Estimate = \( 25000 \times \frac{6}{50} = 3000 \)
### Histograms

**Employee**(*ssn*, *name*, *age*)

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

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

<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>

CSE 544 - Winter 2020
**Histograms**

**Employee**(ssn, name, age)

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

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

<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
Discuss the paper

• Why do they use the IMDB database instead of TPC-H?

• Do cardinality estimators typically under- or over-estimate?

• From cardinality to cost: how critical is that?
## Single Table Estimation

<table>
<thead>
<tr>
<th></th>
<th>median</th>
<th>90th</th>
<th>95th</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL</td>
<td>1.00</td>
<td>2.08</td>
<td>6.10</td>
<td>207</td>
</tr>
<tr>
<td>DBMS A</td>
<td>1.01</td>
<td>1.33</td>
<td>1.98</td>
<td>43.4</td>
</tr>
<tr>
<td>DBMS B</td>
<td>1.00</td>
<td>6.03</td>
<td>30.2</td>
<td>104000</td>
</tr>
<tr>
<td>DBMS C</td>
<td>1.06</td>
<td>1677</td>
<td>5367</td>
<td>20471</td>
</tr>
<tr>
<td>HyPer</td>
<td>1.02</td>
<td>4.47</td>
<td>8.00</td>
<td>2084</td>
</tr>
</tbody>
</table>

**Table 1: Q-errors for base table selections**

Discuss histograms v.s. samples
Single Table Estimation

• 1d Histograms: accurate for selection on a single equality or range predicate; poor for multiple predicates; useless for LIKE

• Samples: great for correlations, or predicates like LIKE; poor for low selectivity predicates: estimate is 0, then use "magic constants"
[How good are they]

Joins (0 to 6)

Figure 3: Quality of cardinality estimates for multi-join queries in comparison with the true cardinalities. Each boxplot summarizes the error distribution of all subexpressions with a particular size (over all queries in the workload)
How good are they

TPC-H v.s. Real Data (IMDB)
Cardinalities to Cost

- Cardinality estimation creates largest errors
- Complex or simple cost models don’t differ much

Their own simple formula

Postgres cost

No I/O, keep only CPU

How good are they
Yet Another Difficulties

- SQL Queries are often issued from applications
- Optimized once using `prepare` statement, executed often
- The constants in the query are not known until execution time: optimized plan may be suboptimal
select
  o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) / sum(volume)
from
(select YEAR(o_orderdate) as o_year,
     l_extendedprice * (1 - l_discount) as volume,
     n2.n_name as nation
from part, supplier, lineitem, orders,
     customer, nation n1, nation n2, region
where p_partkey = l_partkey and s_suppkey = l_suppkey
  and l_orderkey = o_orderkey and o_custkey = c_custkey
  and c_nationkey = n1.n_nationkey
  and n1.n_regionkey = r_regionkey
  and r_name = 'AMERICA'
  and s_nationkey = n2.n_nationkey
  and o_orderdate between '1995-01-01'
  and '1996-12-31'
  and p_type = 'ECONOMY ANODIZED STEEL'
  and s_acctbal ≤ C1 and l_extendedprice ≤ C2 ) as all_nations
group by o_year order by o_year

Optimize without knowing C1, C2
Different optimal plans for different $C_1$, $C_2$
Discussion

• Cardinality estimation = open problem
• Histograms:
  – Small number of buckets (why?)
  – Updated only periodically (why?)
  – No 2d histograms (except db2) why?
• Samples:
  – Fail for low selectivity estimates
  – Useless for joins
• Cross-join correlation – open problem
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 “architecture” paper)
Access Path Selection

Supplier(sid, sname, scity, sstate)
Selection condition: sid > 300 \land scity='Seattle'
Access Path Selection

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

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 = 7/10 * 100 = 70
Access Path Selection

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

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

Which access path should we use?

1. Sequential scan: cost = 100
2. Index scan on sid: cost = \( \frac{7}{10} \times 100 = 70 \)
3. Index scan on scity: cost = \( \frac{1000}{20} = 50 \)
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: $\sigma_{c1}(\sigma_{c2}(R))$ same as $\sigma_{c2}(\sigma_{c1}(R))$
  - Cascading: $\sigma_{c1 \land c2}(R)$ 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 $(R \bowtie S) \bowtie T$
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

$$\gamma_{A, \sum(D)}(R(A, B) \Join_{B=C} S(C, D)) = ?$$

$$R(A, B), S(C, D)$$
Group-by and Join

\[ R(A, B), \ S(C, D) \]

\[
\gamma_A, \ \text{sum}(D) \left( R(A,B) \Join_{B=C} S(C,D) \right) = \\
\gamma_A, \ \text{sum}(D) \left( R(A,B) \Join_{B=C} (\gamma_C, \ \text{sum}(D) S(C,D)) \right)
\]

These are very powerful laws. They were introduced only in the 90’s.
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Key / Foreign-Key

Select x.pno, x.quantity
From Supply x, Supplier y
Where x.sid = y.sid
Key / Foreign-Key

Select x.pno, x.quantity
From Supply x, Supplier y
Where x.sid = y.sid

Select x.pno, x.quantity
From Supply x
Key / Foreign-Key

```
Select x.pno, x.quantity
From Supply x, Supplier y
Where x.sid = y.sid
```

What constraints do we need for correctness?

```
Select x.pno, x.quantity
From Supply x
```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Key / Foreign-Key

Select x.pno, x.quantity
From Supply x, Supplier y
Where x.sid = y.sid

What constraints do we need for correctness?

1. Supplier.sid = key

Select x.pno, x.quantity
From Supply x
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Key / Foreign-Key

Select x.pno, x.quantity
From Supply x, Supplier y
Where x.sid = y.sid

What constraints do we need for correctness?

1. Supplier.sid = key
2. Supply.sid = foreign key

Select x.pno, x.quantity
From Supply x
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Key / Foreign-Key

Select x.pno, x.quantity
From Supply x, Supplier y
Where x.sid = y.sid

What constraints do we need for correctness?

1. Supplier.sid = key
2. Supply.sid = foreign key
3. Supply.sid NOT NULL

Select x.pno, x.quantity
From Supply x
Semi-Join Reduction

Semi-join definition:

\[
R \bowtie S = \Pi_{\text{attr}(R)}(R \bowtie S)
\]

Basic law:

\[
\Pi_{\text{attr}(R)}(R \bowtie S) = \Pi_{\text{attr}(R)}((R \bowtie S) \bowtie S)
\]
Example 1

• Example:

\[ Q = R(A,B) \Join S(B,C) \]
Example 1

• Example:

$$Q = R(A,B) \Join S(B,C)$$

• A semijoin reducer is:

$$R_1(A,B) = R(A,B) \bowtie S(B,C)$$
Example 1

• Example:

\[ Q = R(A,B) \Join S(B,C) \]

• A semijoin reducer is:

\[ R_1(A,B) = R(A,B) \Join S(B,C) \]

• The rewritten query is:

\[ Q = R_1(A,B) \Join S(B,C) \]
Example 2

$Q(y,z,u) = R(‘a’, y), S(y,z), T(z,u), K(u,’b’)$

Semi-join reducer:
Example 2

\[ Q(y,z,u) = R('a', y), S(y,z), T(z,u), K(u,'b') \]

Semi-join reducer:

\[ S'(y,z) : - S(y,z) \bowtie R('a', y) \]
Example 2

\[ Q(y,z,u) = R(\text{'a', y}), S(y,z), T(z,u), K(u,'b') \]

Semi-join reducer:

\[
S'(y,z) :- S(y,z) \bowtie R(\text{'a', y}) \\
T'(z,u) :- T(z,u) \bowtie S'(y,z)
\]
Example 2

Q(y,z,u) = R('a', y), S(y,z), T(z,u), K(u,'b')

Semi-join reducer:

S'(y,z) :- S(y,z) ⋉ R('a', y)
T'(z,u) :- T(z,u) ⋉ S'(y,z)
K'(u) :- K(u,'b') ⋉ T'(z,u)
Example 2

Q(y,z,u) = R(‘a’, y), S(y,z), T(z,u), K(u,’b’)

Semi-join reducer:

S’(y,z) :- S(y,z) ▵ R(‘a’, y)
T’(z,u) :- T(z,u) ▵ S’(y,z)
K’(u) :- K(u,’b’) ▵ T’(z,u)
T”’(z,u) :- T’(z,u) ▵ K’(u)
Example 2

Q(y,z,u) = R('a', y), S(y,z), T(z,u), K(u,'b')

Semi-join reducer:

S'(y,z) :- S(y,z) \bowtie R('a', y)
T'(z,u) :- T(z,u) \bowtie S'(y,z)
K'(u) :- K(u,'b') \bowtie T'(z,u)
T''(z,u) :- T'(z,u) \bowtie K'(u)
S''(y,z) :- S'(y,z) \bowtie T''(z,u)
R''(y) :- R('a',y) \bowtie S''(y,z)
Example 2

\[ Q(y,z,u) = R('a', y), S(y,z), T(z,u), K(u,'b') \]

Semi-join reducer:

\[
\begin{align*}
S'(y,z) & : S(y,z) \bowtie R('a', y) \\
T'(z,u) & : T(z,u) \bowtie S'(y,z) \\
K'(u) & : K(u,'b') \bowtie T'(z,u) \\
T''(z,u) & : T'(z,u) \bowtie K'(u) \\
S''(y,z) & : S'(y,z) \bowtie T''(z,u) \\
R''(y) & : R('a', y) \bowtie S''(y,z)
\end{align*}
\]

Reduced query:

\[ Q(y,z,u) = R''(y), S''(y,z), T''(z,u), K''(u) \]
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
Practice

- Database optimizers typically have a database of rewrite rules
- E.g. SQL Server is rumored to have about 500 rules
- Rules become complex as they need to serve specialized types of queries
Left-Deep Plans and Bushy Plans

Left-deep plan

Bushy plan
Figure 9: Cost distributions for 5 queries and different index configurations. The vertical green lines represent the cost of the optimal plan.
Table 2: Slowdown for restricted tree shapes in comparison to the optimal plan (true cardinalities)
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
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

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 better?
Top-down Partial Plans

\[
\text{SELECT} \ * \\
\text{FROM} \ R, S, T \\
\text{WHERE} \ R.B=S.B \text{ and } S.C=T.C \text{ and } R.A<40
\]

\[
\sigma_{A<40}
\]

\[
\text{SELECT} \ R.A, T.D \\
\text{FROM} \ R, S, T \\
\text{WHERE} \ R.B=S.B \text{ and } S.C=T.C
\]

......
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)
How good are they?

<table>
<thead>
<tr>
<th>Method</th>
<th>PK indexes</th>
<th>PK + FK indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PostgreSQL estimates</td>
<td>true cardinalities</td>
</tr>
<tr>
<td></td>
<td>median 95% max</td>
<td>median 95% max</td>
</tr>
<tr>
<td>Dynamic Programming</td>
<td>1.03 1.85 4.79</td>
<td>1.00 1.00 1.00</td>
</tr>
<tr>
<td>Quickpick-1000</td>
<td>1.05 2.19 7.29</td>
<td>1.00 1.07 1.14</td>
</tr>
<tr>
<td>Greedy Operator Ordering</td>
<td>1.19 2.29 2.36</td>
<td>1.19 1.64 1.97</td>
</tr>
</tbody>
</table>

Table 3: Comparison of exhaustive dynamic programming with the Quickpick-1000 (best of 1000 random plans) and the Greedy Operator Ordering heuristics. All costs are normalized by the optimal plan of that index configuration.
Query Optimization: Conclusions

- Query optimizer = critical part of DBMS
- ”Avoid a very bad plan” instead of “find the optimal plan”
- Size estimation + search space + algo
- Essential:
  - set-at-a-time language
  - order-independent

Next time: asymptotic complexity of query evaluation