DATA516/CSED516
Scalable Data Systems and Algorithms

Lecture 3
Query Optimization, Spark
Announcements

• HW2 is posted and due on Nov. 2\textsuperscript{nd}

• Project proposals due on Oct. 29\textsuperscript{th}

• Review was due today (\textit{How good…?})

• Review of three (!) papers due next week
Quick Recap

• What is data independence?
• What are the ops in the relational algebra?
• What is a logical query plan?
• What is a physical query plan?
• Describe briefly 3 join algorithms
Outline for Today

• Query Optimization
  – How good are they?

• Spark
  – May run out of time, please come to section!
Recap

• Optimizer has three components:
  – Search space
  – Cardinality and cost estimation
  – Plan enumeration algorithms
Recap

- Optimizer has three components:
  - Search space
  - Cardinality and cost estimation
  - Plan enumeration algorithms

- Paper addresses three questions:
  - How good are the cardinality estimators?
  - How important is the cost model?
  - How large does the search space need to be?
Paper Outline

• How good are the **cardinality** estimators?

• How important is the **cost** model?

• How large does the **search space** need to be?
The Job Benchmark

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

• IMDB – popular data on the web, can be imported into any RDBMS with moderate effort

Lesson: you can always import your dataset into RDBMS!
The Job Benchmark

JOB Benchmark: 33 templates, 113 queries
Discuss the difference in class:
• SQL query
• SQL query template (or structure)

Group-by Queries
• None in JOB!
• Important in DS; we’ll discuss them later
Review: Cardinality Estimation

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

What are the statistics on base tables?
Review: Cardinality Estimation

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

What are the statistics on base tables?

- Number of tuples (cardinality) \( T(R) \)
- Number of values in \( R.a \): \( V(R,a) \)
- Histograms (later today)
Review: Cardinality Estimation

What are the four assumptions that database systems do?
Review: Cardinality Estimation

What are the four assumptions that database systems do?

• Uniformity
• Independence
• Containment of values
• Preservation of values
Single Table Estimation

\[ \sigma_{A=c}(R) = \frac{T(R)}{V(R,A)} \]

What assumption does this make?
Single Table Estimation

\[ \sigma_{A=c}(R) = \frac{T(R)}{V(R,A)} \]

What assumption does this make? Uniformity
Single Table Estimation

\[ \sigma_{A=c}(R) = \frac{T(R)}{V(R,A)} \]

What assumption does this make?

Uniformity

<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
Histograms

• T(R), V(R,A) too coarse
• Histogram: separate stats per bucket

• In each bucket store:
  – T(bucket)
  – V(bucket,A)
Employee($ssn$, name, age)

Histograms

$T(Employee) = 25000, \ V(Employee, \ age) = 50$

Estimate $\sigma_{age=48}(Employee) = ?$
Histograms

T(Employee) = 25000, V(Employee, age) = 50

Estimate $\sigma_{age=48}(Employee) = ? = 25000/50 = 500$
Employee(ssn, name, age)

Histograms

$T(\text{Employee}) = 25000, \ V(\text{Employee, age}) = 50$

Estimate $\sigma_{\text{age}=48}(\text{Employee}) = \, ? \quad = \frac{25000}{50} = 500$

<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>$T =$</td>
<td>200</td>
<td>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
<tr>
<td>$V =$</td>
<td>3</td>
<td>10</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Estimate $\sigma_{\text{age}=48}(\text{Employee}) = \, ? $
Employee(ssn, name, age)

Histograms

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

Estimate \( \sigma_{\text{age}=48}(\text{Employee}) = ? \quad = \frac{25000}{50} = 500 \)

<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>( T )</td>
<td>200</td>
<td>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
<tr>
<td>( V )</td>
<td>3</td>
<td>10</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Estimate \( \sigma_{\text{age}=48}(\text{Employee}) = ? \quad = \frac{12000}{6} = 2000 \)
Types of Histograms

- Eq-Width
- Eq-Depth
- Compressed: store outliers separately
- 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><strong>T</strong></td>
<td>200</td>
<td>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
<tr>
<td><strong>V</strong></td>
<td>2</td>
<td>8</td>
<td>10</td>
<td>10</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>
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>T</td>
<td>200</td>
<td>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
<tr>
<td>V</td>
<td>2</td>
<td>8</td>
<td>10</td>
<td>10</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>

## Eq-depth:

<table>
<thead>
<tr>
<th>Age:</th>
<th>0..32</th>
<th>33..41</th>
<th>42-46</th>
<th>47-52</th>
<th>53-58</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>1800</td>
<td>2000</td>
<td>2100</td>
<td>2200</td>
<td>1900</td>
<td>1800</td>
</tr>
<tr>
<td>V</td>
<td>8</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>
**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>T</td>
<td>200</td>
<td>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
<tr>
<td>V</td>
<td>2</td>
<td>8</td>
<td>10</td>
<td>10</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>

### Eq-depth:

<table>
<thead>
<tr>
<th>Age</th>
<th>0..32</th>
<th>33..41</th>
<th>42-46</th>
<th>47-52</th>
<th>53-58</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>1800</td>
<td>2000</td>
<td>2100</td>
<td>2200</td>
<td>1900</td>
<td>1800</td>
</tr>
<tr>
<td>V</td>
<td>8</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

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

• Error:
  \[ \sum_{v \in \text{Domain}(A)} \left( |\sigma_{A=v}(R)| - \text{est}_{Hist}(\sigma_{A=v}(R)) \right)^2 \]

• Bucket boundaries = \( \arg\min_{\text{Hist}}(\text{Error}) \)
• Dynamic programming
• Modern databases systems use V-optimal histograms or some variations
Multiple Predicates

- Independence assumption:
  - Simple
  - But often leads to major underestimates

- Modeling correlations:
  - Solution 1: 2d Histograms
  - Solution 2: use sample from the data
Supplier(sid, sname, scity, sstate)

**Independence Assumption**

\[ T(\text{Supplier}) = 250,000 \]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>2000</td>
<td>8000</td>
<td>50000</td>
<td>120000</td>
<td>65000</td>
<td>5000</td>
</tr>
<tr>
<td>V</td>
<td>50</td>
<td>40</td>
<td>250</td>
<td>300</td>
<td>130</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>125000</td>
<td>80000</td>
<td>45000</td>
</tr>
<tr>
<td>V</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

select * from Supplier
where scity = 'Mountainview'
and sstate = 'CA'
Supplier(sid, sname, scity, sstate)

Independence Assumption

T(Supplier) = 250,000

```
select * from Supplier
where scity = 'Mountainview'
and sstate = 'CA'
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>2000</td>
<td>8000</td>
<td>50000</td>
<td>120000</td>
<td>65000</td>
<td>5000</td>
</tr>
<tr>
<td>V</td>
<td>50</td>
<td>40</td>
<td>250</td>
<td>300</td>
<td>130</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>125000</td>
<td>80000</td>
<td>45000</td>
</tr>
<tr>
<td>V</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

Estimate $\sigma_{\text{scity='Mtv' \land sstate='CA'}}(\text{Supplier}) = ?$
Supplier(sid, sname, scity, sstate)

Independence Assumption

T(Supplier) = 250,000

\[
\text{Select random tuple in Supplier, with probability } \frac{1}{T}
\]

\[
\text{Estimate } \sigma_{\text{scity}='Mtv' \land \text{sstate}='CA'} \text{(Supplier) = ?}
\]

\[
\text{select } * \text{ from Supplier where scity = 'Mountainview' and sstate = 'CA'}
\]
Supplier\((\text{sid, sname, scity, sstate})\)

**Independence Assumption**

\[ T(\text{Supplier}) = 250,000 \]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>2000</td>
<td>8000</td>
<td>50000</td>
<td>120000</td>
<td>65000</td>
<td>5000</td>
</tr>
<tr>
<td>V</td>
<td>50</td>
<td>40</td>
<td>250</td>
<td>300</td>
<td>130</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>125000</td>
<td>80000</td>
<td>45000</td>
</tr>
<tr>
<td>V</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

\[
\text{select * from Supplier where scity = 'Mountainview' and sstate = 'CA'}
\]

**Estimate**

\[
\sigma_{\text{scity='Mtv' \land sstate='CA'}}(\text{Supplier}) = ?
\]

Select random tuple in \text{Supplier}, with probability \(1/T\)

\[
\text{Pr(scity = 'Mtv')} = 
\]
**Independence Assumption**

$T(\text{Supplier}) = 250,000$

**Table:**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>2000</td>
<td>8000</td>
<td>50000</td>
<td>120000</td>
<td>65000</td>
<td>5000</td>
<td>T</td>
<td>125000</td>
<td>80000</td>
<td>45000</td>
</tr>
<tr>
<td>V</td>
<td>50</td>
<td>40</td>
<td>250</td>
<td>300</td>
<td>130</td>
<td>100</td>
<td>V</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

**SQL Query:**

```sql
select * from Supplier
where scity = 'Mountainview'
and sstate = 'CA'
```

**Estimate:**

$\sigma_{\text{sscity='Mtv' \land sstate='CA'}}(\text{Supplier}) = ?$

**Select random tuple in Supplier, with probability $1/T$**

$$\Pr(\text{scity} = 'Mtv') = \Pr(\text{scity} = 'Mtv' \mid \text{scity} \in J..M) \times P(\text{scity} \in J..M)$$
Independence Assumption

T(Supplier) = 250,000

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>2000</td>
<td>8000</td>
<td>50000</td>
<td>120000</td>
<td>65000</td>
<td>5000</td>
</tr>
<tr>
<td>V</td>
<td>50</td>
<td>40</td>
<td>250</td>
<td>300</td>
<td>130</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>125000</td>
<td>80000</td>
<td>45000</td>
</tr>
<tr>
<td>V</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

**Select random tuple in Supplier, with probability 1/T**

Pr(scity = ‘Mtv’) = Pr(scity = ‘Mtv’ | scity ∈ J..M) * P(scity ∈ J..M) = \( \frac{1}{V_{J..M}} \times \frac{T_{J..M}}{T} \)
Supplier(sid, sname, scity, sstate)

Independence Assumption

T(Supplier) = 250,000

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>2000</td>
<td>8000</td>
<td>50000</td>
<td>120000</td>
<td>65000</td>
<td>5000</td>
</tr>
<tr>
<td>V</td>
<td>50</td>
<td>40</td>
<td>250</td>
<td>300</td>
<td>130</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>125000</td>
<td>80000</td>
<td>45000</td>
</tr>
<tr>
<td>V</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

select * from Supplier
where scity = ‘Mountainview’
and sstate = ‘CA’

Estimate \( \sigma_{\text{scity='Mtv' \land sstate='CA'}}(\text{Supplier}) = ? \)

Select random tuple in Supplier, with probability \( 1/T \)

\[
\Pr(\text{scity = 'Mtv'}) = \Pr(\text{scity = 'Mtv' | scity} \in J..M) \cdot \Pr(\text{scity} \in J..M) = \frac{1}{V_{J..M}} \cdot \frac{T_{J..M}}{T} \\
\Pr(\text{sstate = 'CA'}) = 
\]
**Supplier** \((\text{sid}, \text{sname}, \text{scity}, \text{sstate})\)

### Independence Assumption

\[ T(\text{Supplier}) = 250,000 \]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>2000</td>
<td>8000</td>
<td>50000</td>
<td>120000</td>
<td>65000</td>
<td>5000</td>
</tr>
<tr>
<td>V</td>
<td>50</td>
<td>40</td>
<td>250</td>
<td>300</td>
<td>130</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>125000</td>
<td>80000</td>
<td>45000</td>
</tr>
<tr>
<td>V</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

select * from Supplier
where scity = 'Mountainview'
and sstate = 'CA'

Select random tuple in **Supplier**, with probability \(1/T\)

\[
\Pr(\text{scity} = 'Mtv') = \Pr(\text{scity} = 'Mtv' \mid \text{scity} \in J..M) \times P(\text{scity} \in J..M) = 1/V_{J..M} \times T_{J..M}/T
\]

\[
\Pr(\text{sstate} = 'CA') = \Pr(\text{sstate} = 'CA' \mid \text{sstate} \in A..J) \times P(\text{sstate} \in A..J)
\]
Supplier(sid, sname, scity, sstate)

Independence Assumption

T(Supplier) = 250,000

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>2000</td>
<td>8000</td>
<td>50000</td>
<td>120000</td>
<td>65000</td>
<td>5000</td>
</tr>
<tr>
<td>V</td>
<td>50</td>
<td>40</td>
<td>250</td>
<td>300</td>
<td>130</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>125000</td>
<td>80000</td>
<td>45000</td>
</tr>
<tr>
<td>V</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

select * from Supplier
where scity = 'Mountainview'
and sstate = 'CA'

Estimate $\sigma_{\text{scity='Mtv' } \land \text{sstate='CA'}}(\text{Supplier}) = ?$

Select random tuple in Supplier, with probability $1/T$

$\Pr(\text{scity = 'Mtv'}) = \Pr(\text{scity = 'Mtv' } | \text{scity } \in \text{J..M}) \ast P(\text{scity } \in \text{J..M}) = \frac{1}{V_{\text{J..M}}} \ast \frac{T_{\text{J..M}}}{T}$

$\Pr(\text{sstate = 'CA'}) = \Pr(\text{sstate = 'CA' } | \text{sstate } \in \text{A..J}) \ast P(\text{sstate } \in \text{A..J}) = \frac{1}{V_{\text{A..J}}} \ast \frac{T_{\text{A..J}}}{T}$
Supplier(sid, sname, scity, sstate)

**Independence Assumption**

\[ T(\text{Supplier}) = 250,000 \]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>2000</td>
<td>8000</td>
<td>50000</td>
<td>120000</td>
<td>65000</td>
<td>5000</td>
</tr>
<tr>
<td>V</td>
<td>50</td>
<td>40</td>
<td>250</td>
<td>300</td>
<td>130</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>125000</td>
<td>80000</td>
<td>45000</td>
</tr>
<tr>
<td>V</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

Select random tuple in Supplier, with probability \( \frac{1}{T} \)

\[
\Pr(\text{scity} = \text{'Mtv'}) = \Pr(\text{scity} = \text{'Mtv'} | \text{scity} \in J..M) \times P(\text{scity} \in J..M) = \frac{1}{V_{J..M}} \times \frac{T_{J..M}}{T}
\]

\[
\Pr(\text{sstate} = \text{'CA'}) = \Pr(\text{sstate} = \text{'CA'} | \text{sstate} \in A..J) \times P(\text{sstate} \in A..J) = \frac{1}{V_{A..J}} \times \frac{T_{A..J}}{T}
\]

\[
\Pr(\text{scity} = \text{'Mtv'} \land \text{sstate} = \text{'CA'}) = \]
Supplier(sid, sname, scity, sstate)

**Independence Assumption**

\[ T(\text{Supplier}) = 250,000 \]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>2000</td>
<td>8000</td>
<td>50000</td>
<td>120000</td>
<td>65000</td>
<td>5000</td>
</tr>
<tr>
<td>V</td>
<td>50</td>
<td>40</td>
<td>250</td>
<td>300</td>
<td>130</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>125000</td>
<td>80000</td>
<td>45000</td>
</tr>
<tr>
<td>V</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

Select * from Supplier
where scity = 'Mountainview'
and sstate = 'CA'

Select random tuple in Supplier, with probability \(1/T\)

Pr(scity = 'Mtv') = \( \Pr(\text{scity} = 'Mtv' \mid \text{scity} \in J..M) \cdot P(\text{scity} \in J..M) = \frac{1}{V_{J..M}} \cdot \frac{T_{J..M}}{T} \)

Pr(sstate = 'CA') = \( \Pr(\text{sstate} = 'CA' \mid \text{sstate} \in A..J) \cdot P(\text{sstate} \in A..J) = \frac{1}{V_{A..J}} \cdot \frac{T_{A..J}}{T} \)

Pr(scity = 'Mtv' \land sstate = 'CA') = \( \frac{1}{V_{J..M}} \cdot \frac{T_{J..M}}{T} \) \cdot \( \frac{1}{V_{A..J}} \cdot \frac{T_{A..J}}{T} \)

**Independence**
Supplier(sid, sname, scity, sstate)

**Independence Assumption**

\[ T(\text{Supplier}) = 250,000 \]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>2000</td>
<td>8000</td>
<td>50000</td>
<td>120000</td>
<td>65000</td>
<td>5000</td>
</tr>
<tr>
<td>V</td>
<td>50</td>
<td>40</td>
<td>250</td>
<td>300</td>
<td>130</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>125000</td>
<td>80000</td>
<td>45000</td>
</tr>
<tr>
<td>V</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

**Select random tuple in Supplier, with probability 1/T**

\[
\begin{align*}
\Pr(\text{scity} = 'Mtv') &= \Pr(\text{scity} = 'Mtv' | \text{scity} \in J..M) \times P(\text{scity} \in J..M) = 1/V_{J..M} \times T_{J..M}/T, \\
\Pr(\text{sstate} = 'CA') &= \Pr(\text{sstate} = 'CA' | \text{sstate} \in A..J) \times P(\text{sstate} \in A..J) = 1/V_{A..J} \times T_{A..J}/T, \\
\Pr(\text{scity} = 'Mtv' \land \text{sstate} = 'CA') &= (1/V_{J..M} \times T_{J..M}/T) \times (1/V_{A..J} \times T_{A..J}/T),
\end{align*}
\]

**Answer:** \( (1/V_{J..M} \times T_{J..M}/T) \times (1/V_{A..J} \times T_{A..J}/T) \times T = 1/1250 \times 1/40 \times 250000 = 5 \)
**Supplier(sid, sname, scity, sstate)**

**Independence Assumption**

\[ T(\text{Supplier}) = 250,000 \]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>2000</td>
<td>8000</td>
<td>50000</td>
<td>120000</td>
<td>65000</td>
<td>5000</td>
</tr>
<tr>
<td>V</td>
<td>50</td>
<td>40</td>
<td>250</td>
<td>300</td>
<td>130</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>125000</td>
<td>80000</td>
<td>45000</td>
</tr>
<tr>
<td>V</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

Select random tuple in `Supplier`, with probability \(1/T\)

\[
\text{Pr}(\text{scity} = \text{Mtv}') = \text{Pr}(\text{scity} = \text{Mtv}' | \text{scity} \in J..M) \times P(\text{scity} \in J..M) = 1/V_{J..M} \times T_{J..M}/T
\]

\[
\text{Pr}(\text{sstate} = \text{CA}') = \text{Pr}(\text{sstate} = \text{CA}' | \text{sstate} \in A..J) \times P(\text{sstate} \in A..J) = 1/V_{A..J} \times T_{A..J}/T
\]

\[
\text{Pr}(\text{scity} = \text{Mtv}' \land \text{sstate} = \text{CA}') = (1/V_{J..M} \times T_{J..M}/T) \times (1/V_{A..J} \times T_{A..J}/T)
\]

**Answer:** \((1/V_{J..M} \times T_{J..M}/T) \times (1/V_{A..J} \times T_{A..J}/T) \times T = 1/1250 \times 1/40 \times 250000 = 5\)

This is likely an underestimate. Why?
Modeling Correlations

1. Multi-dimensional histograms
   - Also called column-group statistics

2. Sample from the data
Supplier(sid, sname, scity, sstate)

2d-Histogram

T(Supplier) = 250,000

Estimate $\sigma_{\text{scity}=\text{Mtv} \land \text{sstate}=\text{CA}}$(Supplier) = ?
Supplier(sid, sname, scity, sstate)

2d-Histogram

T(Supplier) = 250,000

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>2000</td>
<td>8000</td>
<td>50000</td>
<td>120000</td>
<td>65000</td>
<td>5000</td>
</tr>
<tr>
<td>V</td>
<td>50</td>
<td>40</td>
<td>250</td>
<td>300</td>
<td>130</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>125000</td>
<td>80000</td>
<td>45000</td>
</tr>
<tr>
<td>V</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

Estimate $\sigma_{\text{scity}=\text{Mtv} \land \text{sstate}=\text{CA}}(\text{Supplier}) = ?$
Supplier(sid, sname, scity, sstate)

2d-Histogram

T(Supplier) = 250,000

2d Histogram

1d Histograms

Estimate $\sigma_{\text{scity}='Mtv' \land \text{sstate}='CA'}(Supplier) = \ ?$

Answer: $T_{\text{histogram}} / V_{\text{histogram}}$
Sample

• Compute a small, uniform sample from Supplier

Estimate $\sigma_{\text{sscity='Mtv' \land sstate='CA'}(\text{Supplier}) = ?$
Sample

- Compute a small, uniform sample from Supplier

- Use Thomson’s estimator:

\[ \sigma_{\text{sscity}='Mtv' \land \text{sstate}='CA'}(\text{Supplier}) = ? \]
Sample

- Compute a small, uniform sample from Supplier
- Use Thomson’s estimator:

$$\sigma_{\text{sscity='Mtv' \land sstate='CA'}}(\text{Supplier}) = ?$$

Answer: $$\sigma_{\text{sscity='Mtv' \land sstate='CA'}}(\text{Sample}) \ast \frac{T(\text{Supplier})}{T(\text{Sample})}$$
Correlations

• Solution 1: 2d histograms
  – Plus: can be accurate for 2 predicates
  – Minus: unclear how to use for 3 or more preds
  – Minus: limited number of buckets (why?)
  – Minus: too many 2d histogram candidates

• Solution 2: sampling
  – Plus: can be accurate for >2 predicates
  – Plus: work for complex preds, e.g. “like”
  – Minus: fail for low selectivity predicates
Correlations

• Solution 1: 2d histograms
  – Plus: can be accurate for 2 predicates
  – Minus: unclear how to use for 3 or more preds
  – Minus: limited number of buckets (why?)
  – Minus: too many 2d histogram candidates

• Solution 2: sampling
  – Plus: can be accurate for >2 predicates
  – Plus: work for complex preds, e.g. “like”
  – Minus: fail for low selectivity predicates
Correlations

• Solution 1: 2d histograms
  – Plus: can be accurate for 2 predicates
  – Minus: unclear how to use for 3 or more preds
  – Minus: limited number of buckets (why?)
  – Minus: too many 2d histogram candidates

• Solution 2: sampling
  – Plus: can be accurate for >2 predicates
  – Plus: work for complex preds, e.g. “like”
  – Minus: fail for low selectivity predicates
Correlations

• Solution 1: 2d histograms
  – Plus: can be accurate for 2 predicates
  – Minus: unclear how to use for 3 or more preds
  – Minus: limited number of buckets (why?)
  – Minus: too many 2d histogram candidates

• Solution 2: sampling
  – Plus: can be accurate for >2 predicates
  – Plus: work for complex preds, e.g. “like”
  – Minus: fail for low selectivity predicates
Correlations

• Solution 1: 2d histograms
  – Plus: can be accurate for 2 predicates
  – Minus: unclear how to use for 3 or more preds
  – Minus: limited number of buckets (why?)
  – Minus: too many 2d histogram candidates

• Solution 2: sampling
  – Plus: can be accurate for >2 predicates
  – Plus: work for complex preds, e.g. “like”
  – Minus: fail for low selectivity predicates
Recap: Single Table Estimation

\[ \sigma_{A=c}(R) = \frac{T(R)}{V(R,A)} \]

Assumes uniformity

<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
[How good are they]

Review: Estimate Join Size

Estimate: $T(R \bowtie_{A=B} S) = ??$
Review: Estimate Join Size

Estimate: \( T(R \bowtie_{A=B} S) = ?? \)

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

What assumptions do we make?
Review: Estimate Join Size

Estimate: $T(R \bowtie_{A=B} S) = ??$

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

What assumptions do we make?
• Uniformity
• Containment of values
• Independence:
  • less obvious
  • reason is that both $T(R), T(S)$ are estimated too
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).
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)
Discussion

• Paper explains the need for real data

• Synthetic data used in benchmarks is often generated using uniform, independent distributions; formulas for cardinality estimation are perfect
TPC-H v.s. Real Data (IMDB)

How good are they?
[How good are they]

TPC-H v.s. Real Data (IMDB)

Huge errors

Perfect estimates
Impact of Mis-estimates

- Sec.4 (probably more than you want to know)
- Simple configuration (key index only):
  - Minor performance impact, because the big, “fact” table needs to be scanned anyway
  - Most come from nested-loop joins (why?)
  - Most of the rest come from hash-join (why?)
  - Briefly discuss re-hashing
- More complex configuration
  - Higher perf. Impact

Figure 7: Slowdown of queries using PostgreSQL estimates w.r.t. using true cardinalities (different index configurations)
Paper Outline

• How good are the cardinality estimators?

• How important is the cost model?

• How large does the search space need to be?
Review: Cost Model

Cost model: for each physical operator we use a formula to convert cardinality to cost

• Example: nested loop join $R \bowtie S$
  
  \[ \text{Cost} = c_1 \cdot T(R) + c_2 \cdot T(R) \cdot T(S) \]
Review: Cost Model

Cost model: for each physical operator we use a formula to convert cardinality to cost

- Example: nested loop join $R \bowtie S$
  - Cost = $c_1 \times T(R) + c_2 \times T(R) \times T(S)$

- Example: hash-join $R \bowtie S$
  - Cost = $c_3 \times T(R) + c_4 \times T(S)$  // $c_3 \neq c_4$
Review: Cost Model

Cost model: for each physical operator we use a formula to convert cardinality to cost

• Example: nested loop join $R \bowtie S$
  – Cost = $c_1 \times T(R) + c_2 \times T(R) \times T(S)$

• Example: hash-join $R \bowtie S$
  – Cost = $c_3 \times T(R) + c_4 \times T(S)$ // $c_3 \neq c_4$

• Difficult to choose the right constants!
Review: Cost Model

Cost model: for each physical operator we use a formula to convert cardinality to cost

• Example: nested loop join $R \bowtie S$
  – Cost = $c_1 \cdot T(R) + c_2 \cdot T(R) \cdot T(S)$

• Example: hash-join $R \bowtie S$
  – Cost = $c_3 \cdot T(R) + c_4 \cdot T(S)$  // $c_3 \neq c_4$

• Difficult to choose the right constants!

How important is the cost model?
Cardinalities to Cost
Cardinalities to Cost

Postgres cost
[How good are they]

Cardinalities to Cost

Postgres cost

No I/O, keep only CPU

(a) PostgreSQL estimates
(b) true cardinalities
(c) runtime [ms] [log scale]
(d) tuned cost model
(e) simple cost model
(f) standard cost model
Cardinalities to Cost

Postgres cost

No I/O, keep only CPU

Their own simple formula
Cardinalities to Cost

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

**Postgres cost**

No I/O, keep only CPU

Their own simple formula
SQL Queries issued from applications:

- Query is optimized once: \textit{prepare}
- Then, executed repeatedly

Query constants are unknown until execution: optimized plan is suboptimal
Jayant Haritsa, ICDE’2019 tutorial

```
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
```
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 C1, C2
Paper Outline

• How good are the cardinality estimators?

• How important is the cost model?

• How large does the search space need to be?
Search Space

• The set of alternative plans

• Rewrite rules; examples:
  – Push selections down: $\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$
  – Join reorder: $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
  – Push aggregates down (later today)

• Types of join trees (next)
[How good are they]

The need for a rich search space

**Figure 9:** Cost distributions for 5 queries and different index configurations. The vertical green lines represent the cost of the optimal plan.
Types of Join Trees

• Based on the join condition:
  – With cartesian products
  – Without cartesian products

• Based on the shape:
  – Left deep
  – Right deep
  – Zig-zag
  – Bushy
Cartesian Product: with or without

R(A,B) \bowtie_{R.B=S.B} S(B,C) \bowtie_{S.C=T.C} T(C,D)

Without cartesian product
Cartesian Product: with or without

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

Without cartesian product
Cartesian Product: with or without

$R(A, B) \bowtie_{R.B=S.B} S(B, C) \bowtie_{S.C=T.C} T(C, D)$

Without cartesian product

$R(A, B) \bowtie_{R.B=S.B} S(B, C) \bowtie_{S.C=T.C} T(C, D)$

$R(A, B) \bowtie S(B, C) \bowtie T(C, D)$

$R(A, B) \bowtie S(B, C) \bowtie T(C, D)$

$R(A, B) \bowtie S(B, C) \bowtie T(C, D)$

$R(A, B) \bowtie S(B, C) \bowtie T(C, D)$

$R(A, B) \bowtie S(B, C) \bowtie T(C, D)$
Cartesian Product: with or without

\[ R(A,B) \Join_{B=S.B} S(B,C) \Join_{S.C=T.C} T(C,D) \]

Without cartesian product

With cartesian product
Cartesian Product: with or without

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

Without cartesian product

\[ R(A,B) \bowtie_{S.C=T.C} \bowtie_{R.B=S.B} \]

When could this plan be better?

With cartesian product

\[ R(A,B) \bowtie_{B=B \land C=C} T(C,D) \bowtie_{S.B=T.C} \]

\[ R(A,B) \bowtie \times S(B,C) \bowtie T(C,D) \]
Cartesian Product: with or without

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

**Without cartesian product**

**With cartesian product**

When could this plan be better?

When \( R, T \) are very small, and \( S \) is very large
Shapes of Join Trees

Left deep

R_1  R_2

\ldots

R_{n-1}  R_n

Hash-tables built on right relations
Shapes of Join Trees

Left deep

R₁

R₂

Rₙ⁻¹

Rₙ

⋯

Right deep

R₁

R₂

Rₙ⁻¹

Rₙ

⋯

Hash-tables built on right relations
Shapes of Join Trees

Left deep

R_1

R_2

R_{n-1}

R_n

Right deep

R_1

R_2

R_3

R_{n}

Bushy
Shapes of Join Trees

Left deep

\[ R_n \]
\[ \ldots \]
\[ R_{n-1} \]
\[ R_2 \]
\[ R_1 \]

Bushy

\[ \ldots \ldots \ldots \]

Hash-tables built on right relations

Right deep

\[ R_n \]
\[ \ldots \]
\[ R_3 \]
\[ R_2 \]
\[ R_1 \]

Zig-zag

\[ R_5 \]
\[ \ldots \]
\[ R_4 \]
[How good are they]

Left/right convention switched:
Right-deep build all hash tables first.
Unclear to me why they are worst.

<table>
<thead>
<tr>
<th></th>
<th>PK indexes</th>
<th></th>
<th></th>
<th>PK + FK indexes</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>median</td>
<td>95%</td>
<td>max</td>
<td>median</td>
<td>95%</td>
<td>max</td>
</tr>
<tr>
<td>zig-zag</td>
<td>1.00</td>
<td>1.06</td>
<td>1.33</td>
<td>1.00</td>
<td>1.60</td>
<td>2.54</td>
</tr>
<tr>
<td>left-deep</td>
<td>1.00</td>
<td>1.14</td>
<td>1.63</td>
<td>1.06</td>
<td>2.49</td>
<td>4.50</td>
</tr>
<tr>
<td>right-deep</td>
<td>1.87</td>
<td>4.97</td>
<td>6.80</td>
<td>47.2</td>
<td>30931</td>
<td>738349</td>
</tr>
</tbody>
</table>

Table 2: Slowdown for restricted tree shapes in comparison to the optimal plan (true cardinalities)
Search Space: Discussion

- Search space can be huge

- Database systems often reduce it by applying heuristics:
  - No cartesian products
  - Restrict to left-deep trees (or other restriction)
Rewrite Rules

• We have seen last time:
  – Push selection down: $\sigma_C(R \Join S) = \sigma_C(R) \Join S$
  – AND: $\sigma_{C1 \text{ and } C2}(R \Join S) = \sigma_{C1}($ $\sigma_{C2}(R \Join S))$
  – Join associativity: $(R \Join S) \Join T = R \Join (S \Join T)$
  – Join commutativity: $R \Join S = S \Join R$

• Two more rules
  – Push aggregates down
  – Remove redundant joins

Very important for Data Science!
Motivation

• Try this in Redshift

```sql
select count(*) from customer;
```

Answer: 1500000
Time: 2 s
Motivation

• Try this in Redshift

```
select count(*) from customer;
Answer: 1500000
Time: 2 s
```

```
select count(*) from lineitem;
Answer: 59986052
Time: 1 s
```
Motivation

• Try this in Redshift

select count(*) from customer;
Answer: 1500000
Time: 2 s

select count(*) from lineitem;
Answer: 59986052
Time: 1 s

select count(*) from customer, lineitem;
Motivation

• Try this in Redshift

```sql
select count(*) from customer;
```
Answer: 1500000
Time: 2 s

```sql
select count(*) from lineitem;
```
Answer: 59986052
Time: 1 s

```sql
select count(*) from customer, lineitem;
```
Timeout!!!
Motivation

• Try this in Redshift

select count(*) from customer;
Answer: 1500000
Time: 2 s

select count(*) from lineitem;
Answer: 59986052
Time: 1 s

select count(*) from customer, lineitem;
Timeout!!!

But 3rd query is simply the **product** of the first two!
Pushing Aggregates Down

\[ \mathcal{Y}_{Y,Z,\text{sum}(A*B*C*\cdots)} \]

\[ \bowtie_X \]

\[ \ldots \quad \ldots \]

select Y,Z, sum(A*B*C*...) from...where...
group by Y, Z
Pushing Aggregates Down

As data scientists, you may really need this optimization; do it manually, if needed!

```
select Y, Z, sum(A*B*C*...) from...where...
group by Y, Z
```

\[ \mathcal{Y}_{Y,Z,\text{sum}(A*B*C*\ldots)} \]
Pushing Aggregates Down

As data scientists, you may really need this optimization; do it manually, if needed!
Pushing Aggregates Down

select Y,Z, sum(A*B*C*...) from...where... group by Y, Z

As data scientists, you may really need this optimization; do it manually, if needed!
Pushing Aggregates Down

select Y, Z, sum(A*B*C*...) from...where...
group by Y, Z

As data scientists, you may really need this optimization; do it manually, if needed!
Pushing Aggregates Down

select Y,Z, sum(A*B*C*…) from...where...
group by Y, Z

As data scientists, you may really need this optimization; do it manually, if needed!
Pushing Aggregates Down

select Y, Z, sum(A*B*C*...) from...where...
group by Y, Z

As data scientists, you may really need this optimization; do it manually, if needed!
Pushing Aggregates Down

select Y,Z, sum(A*B*C*...) from...where...
group by Y, Z

Group by Y,Z (again)
multiply the two sums, and sum again

Group by the attrs from the left

Sum only over the attrs from the left

Group by the attrs from the right

Sum only over the attrs from the right

As data scientists, you may really need this optimization; do it manually, if needed!
Example 1

SELECT count(*) from R, S where R.x=S.x
Example 1

SELECT count(*) from R, S where R.x = S.x

R:  | x | y |
---|---|---|
   | b | a |
   | b | c |
   | f | d |
   | h | g |

S:  | x | z |
---|---|---|
   | b | g |
   | b | k |
   | h | m |

Answer = ????
Example 1

SELECT count(*) from R, S where R.x=S.x

Answer = 5
Runtime = O(N^2)
Example 1

SELECT count(*) from R, S where R.x=S.x

Answer = 5

Runtime = O(N^2)
Example 1

```
SELECT count(*) from R, S where
R.x = S.x
```

Answer = 5

Runtime = $O(N^2)$

<table>
<thead>
<tr>
<th>R:</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>f</td>
<td>d</td>
</tr>
<tr>
<td></td>
<td>h</td>
<td>g</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S:</th>
<th>x</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>b</td>
<td>g</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>k</td>
</tr>
<tr>
<td></td>
<td>h</td>
<td>m</td>
</tr>
</tbody>
</table>

A:  

<table>
<thead>
<tr>
<th>x</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>f</td>
<td>1</td>
</tr>
<tr>
<td>h</td>
<td>1</td>
</tr>
</tbody>
</table>

B:  

<table>
<thead>
<tr>
<th>x</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>h</td>
<td>1</td>
</tr>
</tbody>
</table>

A $\bowtie$ B:

<table>
<thead>
<tr>
<th>x</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>h</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$\gamma_{count(*)}$

$\gamma_{sum(c*d)}$
Example 1

```
SELECT count(*) from R, S where R.x=S.x

R: | x  | y |
---|----|---|
 b | a  |
 b | c  |
 f | d  |
 h | g  |

S: | x  | z |
---|----|---|
 b | g  |
 b | k  |
 h | m  |

Answer = 5
Runtime = O(N^2)

A: | x | c |
---|---|---|
 b | 2 |
 f | 1 |
 h | 1 |

B: | x | d |
---|---|---|
 b | 2 |
 h | 1 |

A \ JOIN \ B

Answer = 5
Runtime = O(N)
```
Example 2

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

SELECT x.sstate, sum(y.quantity * z.price)
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
GROUP BY x.sstate
Example 2

\[
\gamma_{x.sstate, \text{sum}(y.quantity*z.price)}
\]

\[
\bowtie_{x.sid = y.sid}
\]

\[
\bowtie_{y.pno = z.pno}
\]

SELECT x.sstate, \text{sum}(y.quantity*z.price)
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
GROUP BY x.sstate
Example 2

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

\[
\gamma_{x.sstate, \text{sum}(y.quantity \cdot z.price)}
\]

\[
\bowtie_{x.sid = y.sid}
\]

\[
\bowtie_{y.pno = z.pno}
\]

Supplier x
Supply y
Part z

\[
\gamma_{y.sid, \text{sum}(y.quantity \cdot z.price)} \rightarrow s
\]

\[
\bowtie_{y.pno = z.pno}
\]

Supplier x
Supply y
Part z

\[
\gamma_{x.sstate, \text{sum}(s)}
\]

\[
\bowtie_{x.sid = y.sid}
\]

SELECT x.sstate, \text{sum}(y.quantity \cdot z.price)
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
GROUP BY x.sstate
Discussion

• Join-aggregates: common in data science
• Implementation in RDBMS seems spotty:
  – Postgres: NO (someone started, abandoned)
  – Redshift: NO (I don’t know the status)
  – SQL Server: YES (at least a few years back)
  – Snowflake: ??
• You may have to force this manually, by writing nested SQL queries
• Let’s make sure we understand it (next)
Redundant Foreign-key / key Joins

- Simple, highly effective

- Almost all engines implement this
Select x.pno, x.quantity
From Supply x, Supplier y
Where x.sid = y.sid
Select x.pno, x.quantity
From Supply x, Supplier y
Where x.sid = y.sid

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

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

Only if these constraints hold:

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

• Database optimizers typically have a database of rewrite rules

• E.g. SQL Server: 400+ rules

• Rules become complex as they need to serve specialized types of queries
Query Optimization

1. Search space

2. Cardinality and cost estimation

3. Plan enumeration algorithms

Discussed already
Two Types of Plan Enumeration Algorithms

• Dynamic programming (in class)
  – Based on System R [Selinger 1979]
  – Join reordering algorithm

• Rule-based algorithm (will not discuss)
  – Database of rules (=algebraic laws)
  – Usually: dynamic programming

• Today’s systems combine both
System R Optimizer

For each subquery $Q \subseteq \{R_1, \ldots, R_n\}$, compute best plan:

- **Step 1:** $Q = \{R_1\}, \{R_2\}, \ldots, \{R_n\}$

- **Step 2:** $Q = \{R_1, R_2\}, \{R_1, R_3\}, \ldots, \{R_{n-1}, R_n\}$

- ...  

- **Step n:** $Q = \{R_1, \ldots, R_n\}$

Avoid cartesian products; possibly restrict tree shapes
Details

For each subquery $Q \subseteq \{R_1, \ldots, R_n\}$ store:

- Estimated Size($Q$)
- A best plan for $Q$: Plan($Q$)
- The cost of that plan: Cost($Q$)
Details

**Step 1**: single relations \{R_1\}, \{R_2\}, \ldots, \{R_n\}

- Size = T(R_i)

- Best plan: scan(R_i)

- Cost = c*T(R_i)  // c=the cost to read one tuple
Details

Step k = 2…n:
For each \( Q = \{R_{i_1}, ..., R_{i_k}\} \) // w/o cartesian product

- Size = estimate the size of Q

- For each j=1,…,k:
  - Let: \( Q' = Q - \{R_{i_j}\} \)
  - Let: \( \text{Plan}(Q') \bowtie R_{i_j} \quad \text{Cost}(Q') + \text{CostOf}(\bowtie) \)

- Plan(Q), Cost(Q) = cheapest of the above
### Is Dynamic Programming needed?

<table>
<thead>
<tr>
<th></th>
<th>PK indexes</th>
<th></th>
<th>PK + FK indexes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PostgreSQL estimates</td>
<td>true cardinalities</td>
<td>PostgreSQL estimates</td>
<td>true cardinalities</td>
</tr>
<tr>
<td></td>
<td>median 95% max</td>
<td>median 95% max</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>
<td>1.66 169 186367</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>
<td>2.52 365 186367</td>
<td>1.02 4.72 32.3</td>
</tr>
<tr>
<td>Greedy Operator Ordering</td>
<td>1.19 2.29 2.36</td>
<td>1.19 1.64 1.97</td>
<td>2.35 169 186367</td>
<td>1.20 5.77 21.0</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.
Discussion

• All database systems implement Selinger’s algorithm for join reorder

• For other operators (group-by, aggregates, difference): rule-based

• Many search strategies beyond dynamic programming
Final Discussion

• Optimizer has three components:
  – Search space
  – Cardinality and cost estimation
  – Plan enumeration algorithms

• Optimizer realizes *physical data independence*

• Weakest link: cardinality estimation
  – Poor plans are almost always due to that
Spark
Motivation

• Limitations of relational database systems:
  – Single server (at least traditionally)
  – SQL is a limited language (eg no iteration)

• Spark:
  – Distributed system
  – Functional language (Java/Scala) good for ML

• Implementation:
  – Extension of MapReduce
  – Distributed physical operators
Review: Single Client

E.g. data analytics
Review: Client-Server

E.g. accounting, banking, …

Connection: ODBC, JDBC
Review: Three-tier connection (ODBC, JDBC)

E.g. Web commerce
Review: Distributed Database

E.g. large-scale analytics or...

App server

ODBC, JDBC

http

Sharded database **Spark**, Snowflake

...social networks
Programming in Spark

• A Spark program consists of:
  – Transformations (map, reduce, join…).  Lazy
  – Actions (count, reduce, save...).  Eager

• **Eager**: operators are executed immediately

• **Lazy**: operators are not executed immediately
  – A operator tree is constructed in memory instead
  – Similar to a relational algebra tree
Collections in Spark

RDD<T> = an RDD collection of type T
• Distributed on many servers, not nested
• Operations are done in parallel
• Recoverable via lineage; more later

Seq<T> = a sequence
• Local to one server, may be nested
• Operations are done sequentially
Example from paper, new syntax

Search logs stored in HDFS

```scala
// First line defines RDD backed by an HDFS file
lines = spark.textFile("hdfs://…")

// Now we create a new RDD from the first one
errors = lines.filter(x -> x.startsWith("Error"))

// Persist the RDD in memory for reuse later
errors.persist()
errors.collect()
errors.filter(x -> x.contains("MySQL")).count()
```
Example from paper, new syntax

Search logs stored in HDFS

// First line defines RDD backed by an HDFS file
lines = spark.textFile("hdfs://...")

// Now we create a new RDD from the first one
errors = lines.filter(x -> x.startsWith("Error"))

// Persist the RDD in memory for reuse later
errors.persist()
errors.collect()
errors.filter(x -> x.contains("MySQL")).count()
Example from paper, new syntax

Search logs stored in HDFS

```scala
// First line defines RDD backed by an HDFS file
lines = spark.textFile("hdfs://...")

// Now we create a new RDD from the first one
errors = lines.filter(x -> x.startsWith("Error"))

// Persist the RDD in memory for reuse later
errors.persist()
errors.collect()
errors.filter(x -> x.contains("MySQL")).count()
```

Transformation: Not executed yet...

Action: triggers execution of entire program
Anonymous Functions

A.k.a. lambda expressions, starting in Java 8

```
errors = lines.filter(x -> x.startsWith("Error"))
```
sqlerrors = spark.textFile("hdfs:////...")
  .filter(x -> x.startsWith("ERROR"))
  .filter(x -> x.contains("sqlite"))
  .collect();
The RDD s:

```
sqlerrors = spark.textFile("hdfs://...")
  .filter(x -> x.startsWith("ERROR"))
  .filter(x -> x.contains("sqlite"))
  .collect();
```
Example

The RDDs:

<table>
<thead>
<tr>
<th>Error…</th>
<th>Warning…</th>
<th>Warning…</th>
<th>Error…</th>
<th>Abort…</th>
<th>Abort…</th>
<th>Error…</th>
<th>Error…</th>
<th>Warning…</th>
<th>Error…</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter(&quot;ERROR&quot;)</td>
<td>filter(&quot;ERROR&quot;)</td>
<td>filter(&quot;ERROR&quot;)</td>
<td>filter(&quot;ERROR&quot;)</td>
<td>filter(&quot;ERROR&quot;)</td>
<td>filter(&quot;ERROR&quot;)</td>
<td>filter(&quot;ERROR&quot;)</td>
<td>filter(&quot;ERROR&quot;)</td>
<td>filter(&quot;ERROR&quot;)</td>
<td>filter(&quot;ERROR&quot;)</td>
</tr>
</tbody>
</table>

\[
\text{sqlerrors} = \text{spark.textFile("hdfs://…")}
\]
\[
\quad .\text{filter}(x \rightarrow x.\text{startsWith("ERROR")})
\]
\[
\quad .\text{filter}(x \rightarrow x.\text{contains("sqlite")})
\]
\[
\quad .\text{collect}();
\]
Example

The RDDs:

```
sqlerrors = spark.textFile("hdfs://...")
    .filter(x -> x.startsWith("ERROR"))
    .filter(x -> x.contains("sqlite"))
    .collect();
```
The RDDs:

```
sqloerrors = spark.textFile("hdfs://...")
  .filter(x -> x.startsWith("ERROR"))
  .filter(x -> x.contains("sqlite"))
  .collect();
```
More on Programming Interface

Large set of pre-defined transformations:
• Map, filter, flatMap, sample, groupByKey, reduceByKey, union, join, cogroup, crossProduct, ...

Small set of pre-defined actions:
• Count, collect, reduce, lookup, and save

Programming interface includes iterations
### Transformations:

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>map(f : T -&gt; U)</code></td>
<td><code>RDD&lt;T&gt;</code></td>
<td><code>RDD&lt;U&gt;</code></td>
</tr>
<tr>
<td><code>flatMap(f : T -&gt; Seq(U))</code></td>
<td><code>RDD&lt;T&gt;</code></td>
<td><code>RDD&lt;U&gt;</code></td>
</tr>
<tr>
<td><code>filter(f : T -&gt; Bool)</code></td>
<td><code>RDD&lt;T&gt;</code></td>
<td><code>RDD&lt;T&gt;</code></td>
</tr>
<tr>
<td><code>groupByKey()</code></td>
<td><code>RDD&lt;(K,V)&gt;</code></td>
<td><code>RDD&lt;(K,Seq[V])&gt;</code></td>
</tr>
<tr>
<td><code>reduceByKey(F : (V,V) -&gt; V)</code></td>
<td><code>RDD&lt;(K,V)&gt;</code></td>
<td><code>RDD&lt;(K,V)&gt;</code></td>
</tr>
<tr>
<td><code>union()</code></td>
<td><code>(RDD&lt;T&gt;, RDD&lt;T&gt;)</code></td>
<td><code>RDD&lt;T&gt;</code></td>
</tr>
<tr>
<td><code>join()</code></td>
<td><code>(RDD&lt;(K,V)&gt;, RDD&lt;(K,W)&gt;)</code></td>
<td><code>RDD&lt;(K, (V,W))&gt;</code></td>
</tr>
<tr>
<td><code>cogroup()</code></td>
<td><code>(RDD&lt;(K,V)&gt;, RDD&lt;(K,W)&gt;)</code></td>
<td><code>RDD&lt;(K, (Seq[V], Seq[W]))&gt;</code></td>
</tr>
<tr>
<td><code>crossProduct()</code></td>
<td><code>(RDD&lt;T&gt;, RDD&lt;U&gt;)</code></td>
<td><code>RDD&lt;(T,U)&gt;</code></td>
</tr>
</tbody>
</table>

### Actions:

<table>
<thead>
<tr>
<th>Action</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>count()</code></td>
<td><code>RDD&lt;T&gt;</code></td>
<td><code>Long</code></td>
</tr>
<tr>
<td><code>collect()</code></td>
<td><code>RDD&lt;T&gt;</code></td>
<td><code>Seq&lt;T&gt;</code></td>
</tr>
<tr>
<td><code>reduce(f : (T,T) -&gt; T)</code></td>
<td><code>RDD&lt;T&gt;</code></td>
<td><code>T</code></td>
</tr>
<tr>
<td><code>save(path : String)</code></td>
<td><code>RDD&lt;T&gt;</code></td>
<td>Outputs RDD to a storage system e.g., HDFS</td>
</tr>
</tbody>
</table>
val points = spark.textFile(...)
  .map(parsePoint).persist()

var w = // random initial vector
for (i <- 1 to ITERATIONS) {
  val gradient = points.map{ p =>
    p.x * (1/(1+exp(-p.y*(w dot p.x))) )-1)*p.y
  }.reduce((a,b) => a+b)

  w -= gradient
}