Datenbanksysteme II: Cost Estimation for Cost-Based Optimization

Ulf Leser
Content of this Lecture

- Cost estimation
- Uniform distribution
- Histograms
- Sampling
Cost Estimation

• Rule-based optimizer
  - Transformations depend only on query, not on the actual data
  - No notion of “plan cost”
    • Cannot differentiate join order
    • Cannot decide on access path selection / index usage

• Cost-based optimizer
  - Estimate the cost of each operation
  - Approached by estimating size of intermediate results

• Cost estimation is important for
  - Choosing cheapest possibility for each single operation
  - Finding cheapest plan for entire query
    • Operations have non-local side-effects, especially order
Example

```
SELECT *
FROM    product p, sales S
WHERE   p.id=s.p_id and
        p.price>100
```

- Assume 3300 products, prices between 0-1000 Euro, 1M sales, index on sales.p_id and product.id
- Assuming uniform distribution
  - Price range is 0-1000 =&gt; selectivity of condition is 9/10
    - Expect 9/10*3300 ~ 3000 products
    - Choose BNL, hash, or sort-merge join (depending on buffer available)
Example

```
SELECT *
FROM product p, sales S
WHERE p.id=s.p_id and p.price>100
```

- **Using histograms**
  - Assume 10 equi-width buckets (explanation later)
  - Selectivity of condition is $\frac{5}{3300} \approx 0.0015$
  - **Choose index-join**: scan p, collect id of selected products, use index on sales.p_id to access sales

- **Note**: We are making another assumption – which?
Cost Estimation

• Cost estimation is bottom-up

• Start by building a model of relations
  - Model should be much smaller than relation
  - Should allow for accurate predictions for all operations
  - Should be consistent – same size estimates for different plans of the same subquery
  - Model should be maintained cheaply when relation changes
  - Model should be generated quickly
  - Models need to be stored and accessed efficiently
  - Example: (count, min, max) for each attribute in each relation

• During query optimization: Need to build models of intermediate results from models of inputs using a formula depending on the creating operation
Certainly wrong. Consider PK/FK constraints

Independence assumption:

\[ 112 \times 2000 / 123456 \approx 2 \]

\textbf{Sel: } \( 1/(98-18)*18 = 22.5\% \)

\begin{itemize}
  \item Name(112, Aare, Mater)
  \item Age(112, 80, 98)
  \item Acc#(112, 1, 123456)
\end{itemize}

\begin{itemize}
  \item Name(500, Aare, Mater)
  \item Age(500, 18, 98)
  \item Acc#(500, 1, 123456)
\end{itemize}

\begin{itemize}
  \item Name(1000, Aare, Zyte)
  \item Age(1000, 18, 98)
  \item Acc#(1000, 1, 123456)
\end{itemize}

\textbf{Sel: } 50\%

\begin{itemize}
  \item \( \sigma \) Name < Mater
\end{itemize}
Types of Models

• **Uniform distribution** of values
  - Very small model (count, max, min), simple to build
  - count = “number of distinct values” better than number of tuples
  - Bad predictions if assumption violated

• **Concrete known “standard” distribution**
  - Normal, Zipf, …
  - Can be characterized by few parameters (mean, stddev, …)
  - Very small model, very accurate
  - Expensive check whether appropriate
  - Only for special cases
Types of Models II

• Describe values by function
  - Might be a very small model, very costly to build in general
  - Very accurate if good fitting function is found
  - Rarely used, since few value distributions can be described by simple functions (names? addresses?)
  - Used for approximations, e.g. wavelet transformations

• Approximation of concrete distribution: Histograms
  - Parameterized size, quite simple to build
  - Independent of underlying distribution
  - Accuracy depends on type and size (and timeliness)
Obtaining Model Parameters

• Exhaustive analysis
  - Too expensive for large relations

• **Sampling**
  - Use a *representative subset* of a relation
    • Choose subset at random
    • Not so easy to chose a truly representative samples
  - Accuracy depends on sampling method and size (and timeliness)
  - Examples later
Content of this Lecture

- Cost estimation
- Uniform distribution
- Histograms
- Sampling
Rules of Thumb

• For relation R and attribute A, let
  - \( V(R, A) \) be number of distinct values of A
  - \( \max(R, A), \min(R, A) \) be the maximal/minimal value of A
    • Values that do exist “now”, not maximal / minimal possible values
  - \(|R|\) be the number of tuples in R
  - Note: R may be an intermediate result

• In the following, S denotes the result of a (unary, binary) operation

• Definition
  - The selectivity of a relational operation is the fraction of tuples of the input that will be in the output
Size of a Selection

- We assume $\text{min} \leq \text{const} \leq \text{max}$
- Selection of the form “$A=\text{const}$”
  - $|S| = |R| / v(R,A)$
  - $v(S,A) = 1; \text{max}(S,A)=\text{min}(S,A)=\text{const}$
- Selection of the form “$A < \text{const}$” (or “$A \leq \geq > \text{const}$”)
  - $|S| = |R| / (\text{max-min}) \ast (\text{const-min})$
  - $v(S,A) = v(R,A) / (\text{max-min}) \ast (\text{const-min})$
  - $\text{min}(S,A) = \text{min}; \text{max}(S,A) = \text{const}$
  - Alternative: $|S| = |R| / k$ (e.g. $k=10,15,\ldots$)
    - Idea: With such queries, one usually searches for outliers
    - Very rough estimate, but requires no knowledge of $R$ at all
Size of a Selection II

- **Selection of the form “A \neq \text{const}”**
  - \(|S| = |R| * (v(R,A)-1)/v(R,A)\)
  - \(v(S,A)=v(R,A)\)
  - \(\min(S,A)=\min, \ max(S,A)=\max\)
  - Alternative: \(|S| = |R|\)
Complex Selections

• Selection of the form “Aθc₁ ∧ Bθc₂ ∧ …”
  - Assumption: Statistical independence of values
  - Total selectivity is sel(c₁) * sel(c₂) * …
  - v, min, max are adapted iteratively for each single condition

• Selection of the form “Aθc₁ ∨ Bθc₂ ∨ …”
  - Rephrase into ¬ (¬(Aθc₁) ∧ ¬(Bθc₂) ∧ …)
  - Selectivity is 1- (1-sel(c₁))*(1-sel(c₂))*…

• Selectivity of A=10 ∧ A>10 ?
Projection and Distinct

- **Selectivity of distinct**
  - $|S| = v(R,A)$
  - $v(S,A) = v(R,A)$, $\min(S,A) = \min$, $\max(S,A) = \max$

- **Selectivity of projection**
  - Is 1 under **BAG semantics**
  - Is as selectivity of distinct under **SET semantics**
  - **Caution**
    - In real life, we need to estimate the size of the intermediate relation
    - This requires **number of tuples and size of tuples**
    - We ignore(d) this issue
Projection and Distinct

- Selectivity of grouping
  - Same as selectivity of distinct on group attributes
- Selectivity of `SELECT DISTINCT A, B, C FROM ...`
Projection and Distinct

- **Selectivity of grouping**
  - Same as selectivity of `distinct on group` attributes

- **Selectivity of `SELECT DISTINCT A,B,C FROM ...`**
  - Not easy
  - Clearly, $0 < |S| < v(R,A) \times v(R,B) \times v(R,C)$
  - Suggestion: $|S| = \min(\frac{1}{2} \times |R|, v(R,A) \times v(R,B) \times v(R,C))$

- **Alternative**
  - Multi-dimensional histograms (later)
Selectivity of Joins

- Consider join \( R \bowtie_A T \) (or \( \sigma_{R.A=T.A} (R \times T) \))
- Size of product is \(|R| \times |T|\), but selectivity of the condition?
  - Same problem as for \( \ldots \text{DISTINCT } A, B, C \ldots \)
- Suggestions
  - We assume that joins always are over PK / FK constraints
    - Thus, if \( v(R,A) < v(T,A) \), \( T.A \) is PK in \( T \) and \( R.A \) is FK
    - Each tuple from \( R \) will have \(|T|/v(T,A)\) joining tuples in \( T \)
  - We assume that value sets are similar
    - Thus, a given tuple from \( T \) has a \( 1/v(R,A) \) chance to join with a randomly chosen tuple from \( R \)
    - Symmetrically, a given tuple from \( R \) has a \( 1/v(T,A) \) chance to join with a randomly chosen tuple from \( T \)
    - We perform \(|R| \times |T|\) such experiments
    - Together: \(|S| = |R| \times |T| \times 1/(\max(v(T,A), v(R,A)))\)
Selectivity of Joins

• Together
  - $|S| = |R| \times |T| / \max( v(R,A), v(T,A))$
  - $|R| < |T|$: $v(S,A) = v(R,A)$, $\min(S,A) = \min(R,A)$, $\max(S,A) = \max(R,A)$

• What about $R \bowtie_{R.A < T.B} T$?
Content of this Lecture

- Cost estimation
- Uniform distribution
- Histograms
- Sampling
Histograms

- **Real data** is rarely uniformly distributed
  - Nor Poisson, Gauss, Zipf, ...

- **Solution: Histograms for single attributes**
  - Partition the existing value range into buckets
  - Count frequency of tuples in each bucket (i.e. range)
  - During cost estimation, approximate frequency of a single value in a bucket by the average over all values in this bucket
    - i.e., make uniform distribution assumption inside each bucket

- **Advantage**
  - Lower errors due to smaller ranges for dist. assumption
  - Hope: Frequencies vary less inside smaller ranges
  - Histograms do not help against completely erratic distributions
    - Erratic does not mean random - so what?
Issues

• We must think about
  - How should we chose the \textit{borders of buckets}? 
  - What do we \textit{store for each bucket} (could be more than count)? 
  - How do we \textit{keep buckets up-to-date}?
• Assume normal distribution of weights
  - Spread: 120-40=80, mean: 80, stddev: 12; 100000 people
• Uniform distribution: 100000/80=1250 for each possible weight
• Leads to large errors in almost all possible query ranges
Equi-Width Histograms

- Fix $n$# of buckets
- Borders are equi-distant (border values need not be stored)
- In each bucket, assume average frequency inside bucket
Equi-Width Histograms 2

- Bucket counts can be computed by scanning relation once
- Remaining error depends on
  - Number of buckets (more buckets -> less errors, but more work)
  - Distribution of values in each bucket
Equi-Depth

• Chose borders such that total frequency in each bucket is app. equal
  - Problem if one value is more frequent than $|R|/|\text{buckets}|$ - use / combine with other types of histograms (later)
Equi-Depth

- Buckets have varying size (borders need to be stored)
- Better **fit to data**
- Computation?
  - Sort all values, then jump in equal steps
Example

- **Query**: Number of people with weight between 65-70 (incl)
  - **Real value**: 11603
  - **Uniform distribution**: \((70-65+1)\times1250 = 7500\)
    - **Error**: 4103 \(\approx 35\%\)
  - **Equi-width histogram**
    - Range 60-69 has average 1469
    - Range 70-79 has average 2926
    - **Estimation**: \(5\times1469 + 1\times2926 = 10271\)
      - **Error**: 1332 \(\approx 11\%\)
Example cont’d

- **Query**: Number of people with weight between 65-70 (incl)
  - **Real value**: 11603
  - **Uniform distribution**: \((70-65+1) \times 1250 = 7500\)
    - **Error**: 4103 ~ 35%
  - **Equi-depth histogram**
    - Range 65-69 has average 1850
    - Range 70-73 has average 2581
    - **Estimation**: \(5 \times 1850 + 1 \times 2581 = 11831\)
    - **Error**: 228 ~ 2%

- **Error depends on concrete value or range**
- In general, **equi-depth histograms are considered as better than equi-width histograms**
Other Types of Histograms

- **Serial histograms**
  - Sort values by frequency and build buckets as ranges of frequencies (rare values, less rare values, ...)
  - Frequency ranges of different buckets do not overlap
  - Better fit, but values in buckets must be stored explicitly
    - There are no borders nor consecutive ranges

- **V-optimal histograms**
  - Sort values by frequency and build buckets such that weighted variance is minimized in each bucket
    - Explicitly considers the expected future error
  - **Provably best class of histograms** for “average” queries
    - But costly generation and maintenance
    - Best known algorithm is $O(b*n^2)$ ($n$: n# values, $b$: n# buckets)
Other Types of Histograms

• End-biased histograms
  - Sort values by frequency and build **singleton buckets for largest / smallest frequencies** plus one bucket for all other values
  - Simple form of serial histograms, quite effective for many real-world data distributions (e.g. Zipf-like distributions)

• Commercial systems seem mostly to use **equi-depth and compressed histograms** (mixture of equi-depth and end-biased histograms)

Ioannidis, Y. (2003). "The history of histograms (abridged)". VLDB
Histograms for Join Estimation

- Assume 20K sales and 380 reclamations
  - And a *slightly strange query*, not passing along PK/FK constraints
  - Probably a mistake? But the DB must execute (and optimize) it anyway

```
SELECT  count(*)
FROM    sales S, reclamation R
WHERE   S.productID=R.productID;
```
Example without Histograms

- Without histograms, assuming **uniform distribution**
  - Recall join-formula
  - Gives $|S| \times |R| / (\text{max} (v(R,\text{productID}), v(S,\text{productID}))) \approx 2500$

```
SALES
  salesID
  productID
  ...

RECLAIM
  salesID
  productID
  ...
```

- 20K tuples
- 3K different values
- 380 tuples
- 250 different values
Example with Histograms

- Uniform distribution within buckets
  - \((7000 \times 300/500) + (450 \times 60/500) + \ldots\) ~ 4200
- More complicated if bucket borders do not coincide
- Improvement if numbers of distinct value per bucket are known

<table>
<thead>
<tr>
<th>Range</th>
<th>B.pl D</th>
<th>R.pl D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-499</td>
<td>7000</td>
<td>300</td>
</tr>
<tr>
<td>-999</td>
<td>450</td>
<td>60</td>
</tr>
<tr>
<td>-1499</td>
<td>2650</td>
<td>0</td>
</tr>
<tr>
<td>-1999</td>
<td>5000</td>
<td>0</td>
</tr>
<tr>
<td>-2499</td>
<td>100</td>
<td>20</td>
</tr>
<tr>
<td>-2999</td>
<td>4900</td>
<td>0</td>
</tr>
</tbody>
</table>
Histograms and Complex Conditions

• We only considered histograms for single attributes
• Does not help to estimate selectivity of complex conditions
  - People with weight<30 and age<25 ?
  - People with income>1M and tax depth<500K ?
  - We need to know conditional distributions
  - Clearly, there is a combinatorial explosion in the number of combinations to consider
    • Also: Could be connected by AND, OR, AND NOT, …

• Multidimensional histograms
  - Very active research area
  - Need sophisticated storage structures - multidimensional indexes
Building Histograms

- Usually, computing histograms requires **scanning a table**
  - Potentially for each attribute
- **Cannot be done before each query** – offline statistics
- Indexes can help
  - Statistics such as min, max are directly obtainable
  - Inner nodes of B+ trees ~ equi-depth histograms
  - But we rarely have indexes on all attributes of a relation
Maintaining Histograms

• Idea: Compute once and maintain

• Equi-width histograms
  - Simple; increase/ decrease total frequency in bucket upon insert/delete/update

• Equi-depth histograms
  - Changes in data may influence borders of buckets
  - Option 1: Proceed as for equi-width, accept intermediate inequalities in bucket frequencies
    • ... and regularly re-compute entire histogram
  - Option 2: Implement complex bucket merging/splitting procedures
Maintaining Histograms on Request

- **Compute only on user request**
  - Administrator needs to trigger re-computation of (all, table-wise, attribute-wise, ...) statistics from time to time
  - Otherwise, query performance may degrade
  - Both cases (new or outdated statistics) may lead to unpredictable changes in query behavior
    - To prevent, Oracle provides “query outlines”
- **Automatically maintaining statistics** is a very active research topic
  - General trend: Reduce total cost of ownership
  - Self-optimizing, self-maintaining, zero-administration, ...
Content of this Lecture

- Cost estimation
- Uniform distribution
- Histograms
- Sampling
Sampling

• Scanning a table for computing a histogram is expensive
• But we actually don’t really need all values – we only estimate the distribution (histograms are estimates anyway)
• Solution: Use a sample of the data
  – If chosen randomly, sample should have the same distribution as full data set
  – Very effective: Usually, a 10% sample suffices
• Also useful for approximate COUNT, AVG, SUM, etc.
  – Approximate query processing: Much faster answers in much less time with minimal error
  – Requires estimation of maximal error (confidence values)
  – Again: Very active research area (“Taming the terabyte”)

Problems with Sampling

- How we get a random, 10% sample?
- Reading first 10% of rows is a very bad idea
- Reading a row from 10% of the blocks is about as slow as reading the entire table (sequential reads!)
- Option: Reservoir sampling: Explicitly store and maintain a sample
- Sampling is a build-in database operator; impossible to emulate efficiently