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 and schema information*, but not on the actual data
  - No notion of "cost"
    - Cannot differentiate join order
    - Cannot decide on access path selection / index usage …

- **Cost-based optimizer**
  - Estimate the *cost of each operation* in a QEP
  - Approached by estimating size of intermediate results

- **Cost estimation required for**
  - Choosing best implementation for each operation
  - Finding best 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 => 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 buckets
  - We infer: Selectivity of condition is 5/3300 ~ 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?
  - Maybe people mostly buy expensive goods?
Cost Estimation

• We approach cost estimation bottom-up
• Start by building a model of relations
  - Model should be much smaller than relation
  - Should allow for accurate predictions for all possible operations
    • Selection, projection, group-by, ...
    • We will have to make some compromises
  - Should be consistent – same estimates for different ways of implementing the same subquery
  - Model should be easy to maintain when data changes
  - Model should be generated quickly
  - Models need to be stored and accessed efficiently
  - Models must be easily creatable (better: derivable) for intermediate relations during query processing
Example

- Simple approach: count for each relation, (min, max) for each attribute in each relation
  - Generation requires only one pass
    - Beware: Count usually cannot be derived from used space
  - Data inserts possible in constant time
    - Update/delete: Exact models may require finding new min / max
    - Alternative: Ignore update/delete, accept errors
  - Storing requires only a few bytes per attribute
    - More for string attributes
    - Need not always be exact: “zz” instead of “zweifel”, E3 instead of 975
  - Estimating effect of join not easy, other operations are easy
Certainly wrong. Consider PK/FK constraints

```
Name(112, Aare, Mater)
  Age(112, 80, 98)
  Acc#(112, 1, 123456)
```

```
Name(500, Aare, Mater)
  Age(500, 18, 98)
  Acc#(500, 1, 123456)
```

```
Name(1000, Aare, Zyte)
  Age(1000, 18, 98)
  Acc#(1000, 1, 123456)
```

```
Acc#(2, 1, 123456)
```

```
Sel: \text{1/ (98-18)}*18= 22,5\%
```

```
Sel: 1/(98-18)*18= 22,5\%
```

```
Independence assumption: 112*2000/123456\sim 2
```

```
SELECT C.name, A.balance
FROM customer C, account A
WHERE C.acc# = A.acc# AND
  C.name < "Mater" AND
  C.age > 80
```
Types of Models for One Attribute

• Option 1: **Uniform distribution** of values and **statistical independence** of different attributes
  - Very small model (e.g. count, max, min), simple to build
  - Simple improvement: Also store number of distinct values
  - Arbitrarily bad predictions if assumption violated
  - Cannot capture **correlated attributes**

  • SELECT C.name, C.address FROM customer C, account A WHERE C.acc# = A.acc# AND C.age<19 AND A.balance>100.000
Types of Models II

- Option 2: Known **standard distribution**
  - Normal, Poisson, Zipf, ...
  - Can be characterized by **few parameters** (mean, stddev, ...)
  - Very small model, can be very accurate
    - Weight of persons, number of sales per product, ...
  - But: How should the DB know which distribution is the right one?
    - Must be **specified by developer**
  - Often difficult to propagate through query plans
    - Normal distribution after SELECT is not normal anymore
  - Only used for **special cases**
Types of Models III

- Option 3: Approximation of concrete distribution by histograms
  - Parameterized size, quite simple to build
  - Independent of underlying distribution
  - Accuracy depends on type and size (and timeliness)
Obtaining Model Parameters

- Exhaustive analysis
  - Even $O(|R|)$ might be too expensive for very large relations

- Sampling
  - Use a representative subset of tuples of a relation
    - Choose subset at random
    - Not so easy to choose a truly random samples
  - Accuracy depends on sampling method and size (and timeliness)
  - Examples later
Important Note

• Derived estimations need not be exact
  - Should only help to discern good transformations from bad ones
  - Order of alternatives matter, not concrete cost
• Estimates are often very bad (orders of magnitude)
  - Especially when data deviates from assumptions of the model
  - Still, resulting plans might be very good
• Trade-off: Accuracy of model-derived estimates versus effort to maintain models
Content of this Lecture

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

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

• We discuss impact of each relational operation on parameters of a simple model assuming uniform distributions
  - S will denote the result of a (unary, binary) operation

• For relation R and attribute A, our model consists of
  - 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
Size after a Selection

- We assume $\min \leq \text{const} \leq \max$
- Selection of the form “$A=\text{const}$”
  - $|S| = |R| / v(R,A)$
  - $v(S,A) = 1; \max(S,A) = \min(S,A) = \text{const}$
- Selection of the form “$A<\text{const}$” (or “$A \leq \geq > \text{const}$”)
  - $|S| = |R| / (\max-\min) \times (\text{const}-\min)$
  - $v(S,A) = v(R,A) / (\max-\min) \times (\text{const}-\min)$
  - $\min(S,A) = \min; \max(S,A) = \text{const}$
  - 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 values in $A$ at all
Size of a Selection II

- **Selection of the form “A≠const”**
  - \(|S| = |R| \times \frac{(v(R,A)-1)}{v(R,A)}\)
    - We assume that const exists as value in A
  - \(v(S,A)=v(R,A)\)
    - But we don’t know! Be careful
  - \(\min(S,A)=\min, \max(S,A)=\max\)
  - Alternative: \(|S| = |R|\)
Complex Selections

- **Selection of the form** “$A \theta c_1 \land B \theta c_2 \land \ldots$”
  - Assumption: Statistical independence of values
  - Total selectivity is $\text{sel}(c_1) \times \text{sel}(c_2) \times \ldots$
  - $v, \min, \max$ are adapted iteratively for each single condition

- **Selection of the form** “$A \theta c_1 \lor B \theta c_2 \lor \ldots$”
  - Rephrase into $\neg (\neg (A \theta c_1) \land \neg (B \theta c_2) \land \ldots)$
  - Selectivity is $1 - (1 - \text{sel}(c_1)) \times (1 - \text{sel}(c_2)) \times \ldots$

- **Selectivity of $A=10 \land A>10$ ?**
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 same as selectivity of distinct under **SET semantics**
  - **Caution**
    - In real life, we need to estimate the size of the intermediate relation in bytes
    - This requires **number of tuples and size of tuples**
    - We ignore(d) this issue
DISTINCT and GROUP-BY

- 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: We need to know *correlations of values*
  - Clearly, $0 < |S| < v(R,A) \times v(R,B) \times v(R,C)$
  - Suggestion: $|S| = \min\left( \frac{1}{2}|R|, v(R,A)\times v(R,B)\times v(R,C) \right)$

- **Alternative**
  - Multi-dimensional histograms (later)
  - Note: A, B here may have completely *different domains*, in a join the domains of the joined attributes must be the same
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 join?
  - Need to know about correlations of values in different relations
  - Similar problem as for \ldots DISTINCT A,B,C \ldots,
- Suggestions
  - Option 1: We join a PK with a FK
    - Thus, if $v(R,A)<v(T,A)$, T.A is PK in T and R.A is FK
      - Or vice versa
    - Each FK "finds" its PK
    - Thus: $|S|=|R|$, $\max(S,A)=\max(R,A)$, $\min(S,A)=\min(R,A)$, $v(S,A)=v(R,A)$
Selectivity of Joins

Option 2: Assume that value sets are similar
- Assumption: Users don’t join independent attributes
- Thus, most (all) tuples will find a join partner
- Thus, each tuple from T will join with app. $|R|/v(R,A)$ tuples from R
- Symmetrically, each tuple from R will join with app. $|T|/v(T,A)$ tuples from T
- Thus, we expect $|T|*|R|/v(R,A)$ or $|R|*|T|/v(T,A)$
- Typical solution: $|S| = |R|*|T| / (\max(v(T,A), v(R,A)))$
- $|R| < |T|$: $v(S,A) = v(R,A)$, $\min(S,A) = \min(R,A)$, $\max(S,A) = \max(R,A)$
- Can (and should) be refined by also considering value ranges

What about $R \bowtie_{R.A < T.B} T$?
- For each value $T.B$, estimate which fraction of $R$ has smaller values in $R.A$
Remarks

- We did not discuss effects on other attributes: Home work
  - For instance: Assuming statistical independence, a condition "age<19" does not change min(R,name) or max(R,name)
  - But: "SELECT name, sum(price) as x FROM products GROUP BY product.name" yields v(S,name)=v(P,name), but introduces a new column x whose model must be estimated
Content of this Lecture

- Cost estimation
- Uniform distribution
- Histograms
- Sampling
Histograms

- **Real data** is rarely uniformly distributed
  - Nor Poisson, normal, Zipf, …

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

- **Advantage**
  - **Lower errors** due to smaller ranges for uniformity assumption
  - Hope: Frequencies **vary less inside smaller ranges**
  - Histograms do not help in case of extremely distorted distributions
Issues

• We must think about
  - How should we chose the borders of buckets?
  - What do we store for each bucket (could be more than count)?
  - How do we 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 number of buckets
- Borders are \textit{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 space)
  - Distribution of values in each bucket
Equi-Depth

- Fix number $b$ of buckets
- Chose borders such that frequency of values in each bucket is approximately equal
  - If one value more frequent than $|R|/b$ - use other histograms
Equi-Depth

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

• **Query:** Number of people with weight in [65-70]
  
  - **Real value:** 11603
  
  - **Uniform distribution:** \((70-65+1) \times 1250 = 7500\)
    
    - Error: 4103 ~ 35%
  
  - **Equi-width histogram**
    
    - Range 60-69 has average 1469
    
    - Range 70-79 has average 2926
    
    - Estimation: \(5 \times 1469 + 1 \times 2926 = 10271\)
      
      - Error: 1332 ~ 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: $4103 \sim 35\%$
  - Equi-depth histogram
    • Range 65-69 has average 1850
    • Range 70-73 has average 2581
    • Estimation: $5 \times 1850 + 1 \times 2581 = 11831$
    • Error: $228 \sim 2\%$

• Error depends on concrete value or range

• In general, equi-depth histograms are considered more accurate than equi-width histograms
  - But more costly to build and maintain
Other: 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 consecutive ranges any more
- Range queries must find their values in all buckets

<table>
<thead>
<tr>
<th>Value</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cnt</td>
<td>12</td>
<td>92</td>
<td>10</td>
<td>180</td>
<td>22</td>
<td>20</td>
<td>80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bucket</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values</td>
<td>4</td>
<td>2,5,7</td>
<td>1,3,6</td>
</tr>
<tr>
<td>Total cnt</td>
<td>180</td>
<td>194</td>
<td>42</td>
</tr>
<tr>
<td>$\sigma^2$</td>
<td>0</td>
<td>$\sim1400$</td>
<td>$\sim28$</td>
</tr>
</tbody>
</table>
Other: V-Optimal Histograms

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

<table>
<thead>
<tr>
<th>Value</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cnt</td>
<td>12</td>
<td>92</td>
<td>10</td>
<td>180</td>
<td>22</td>
<td>20</td>
<td>80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bucket</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values</td>
<td>4</td>
<td>2,5</td>
<td>1,3,6,7</td>
</tr>
<tr>
<td>Total cnt</td>
<td>180</td>
<td>172</td>
<td>64</td>
</tr>
<tr>
<td>$\sigma^2$</td>
<td>0</td>
<td>$\sim72$</td>
<td>$\sim35$</td>
</tr>
</tbody>
</table>
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
Content of this Lecture

- Cost estimation
- Uniform distribution
- Histograms
  - Types of histograms
  - Joins, construction, maintenance
- Sampling
Histograms for Join Estimation

- Assume sales and 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 $|S| \times |R| / \left( \max (v(R, \text{productID}), v(S, \text{productID})) \right) \sim 2500$
Example with Histograms

- Uniform distribution within buckets
  - And uniform distribution of distinct values
    - Better: Store cnt of distinct value per bucket
    - \((7000 \times 300/500) + (450 \times 60/500) + \ldots \sim 4200\)
- More complicated if bucket borders do not coincide
  - Which usually is the case for equi-depth histograms
Histograms and Complex Conditions

- We only considered histograms for single attributes
- How to apply for complex conditions?
  - People with weight<30 and age<25?
  - People with income>1M and tax depth<500K?
  - Until now, we assumed statistical independence of attributes
  - Better estimates require conditional distributions
  - But: Combinatorial explosion of the number of combinations
    • Plus: Could be connected by AND, OR, AND NOT, ...
- Multidimensional histograms
  - 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 from a B+ index
  - 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
  - Assumption: Number of buckets and min/max does not change
  - Then everything is simple; increase/ decrease frequencies in bucket upon insert/delete/update
  - (Gross) Changes in min/max: Rebuild histogram

• 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 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 only need to estimate the distribution
  - Histograms are estimates anyway
- Solution: Use a sample of the data
  - If chosen randomly, sample should have same distribution as full data set
  - For large data sets, usually, a 1-10% 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 do 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 must be a build-in database operator; impossible to emulate efficiently