Datenbanksysteme II:
Cost Estimation for Cost-Based Optimization

Ulf Leser
Content of this Lecture

- Cost estimation
- Uniform distribution
- Histograms
- Sampling
- Example: Oracle
- Some empirical observations
Motivation: Cost-based Optimizers

• Find best plan based on estimation of a plan’s cost
• Requires a cost model: How do we compute the cost of an operation, given its input, its output, and its internal computation?
• Most prominent: Size of intermediate results
  – Which are output of some operation and input to other operations
  – This is typically 1:1, for joins 2:1
  – Also called “cardinality estimation”
• In this lecture, we focus on cardinality estimation
  – For good reasons: Probably largest impact
Other Costs

- **Width** of tuples
  - Typically easy to estimate – we’ll mostly skip this

- **Real data access**
  - Disk or memory (or network)
  - Blocked / tuple, random / sequential

- **Computing the predicate**
  - Mostly very cheap: Comparisons
  - But: Aggregations, projections with functions
  - Very expensive: Median
  - Very expensive: Window Functions
Example

- Assume we store for each attribute: (count, min, max)
- 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

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

- Approaching real selectivity: Using **histograms**
  - Assume 10 buckets for price of products
  - We infer: Selectivity of condition is $5/3300 \sim 0.0015$
  - **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 individual 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
  – Should be easy to maintain when data changes
  – Should be generated quickly
  – Needs to be stored and accessed efficiently
  – Should be easily derivable for intermediate relations during query processing
First Model: Uniform Distribution

- With uniform distribution, we only need \((\text{count, min, max})\):
  - “Smaller”: Storing requires only a few bytes per attribute
    - More for string attributes
    - Need not always be exact: “zz” instead of “zweifel”, 5 instead of 5,231
  - “Accurate”: Let’s see (this lecture)
  - “Consistent”: No
  - “Maintainable”: In constant time for INSERT
    - Update/delete: Exact models may require finding new min / max
    - Alternative: Ignore update/delete, accept errors
  - “Fast generation”: Requires only one pass
    - Beware: Count usually cannot be derived from used space
  - “Efficient storage and retrieval”: Small is always efficient
  - “Derivable”: Let’s see (this lecture)
Other Models

- **Recall**: Small, accurate, updateable, derivable
- **Option 2**: Assume one of the *standard distributions*
  - Normal, Poisson, Zipf, ...
    - Weight of persons, number of sales per product, ...
  - Small: Very small model
    - Can be characterized by *few parameters* (mean, stddev, ...)
  - Accurate: Very accurate if values follow distribution tightly
    - But: How should the DB know which distribution is the right one?
      - Must be *specified by developer*
  - Updatable: There are no updates once parameters are known
  - Derivable: Very difficult to *impossible*
    - Normal distribution after SELECT is not normal anymore
    - We cannot use option 2 everywhere in the plan
  - Only used for *special cases*
Other Models II

- **Recall**: Small, accurate, updateable, derivable
- **Option 3**: Approximation of distribution by histograms
  - Different types, more or less adequate for different distributions
  - **Parameterized size**, quite simple to build
  - Accuracy depends on type and size
  - Rather efficient means for updates and derivations
  - Later this lecture
- **Option 4**: Sampling
  - Maintain a random sample of tuples for each relation
  - Estimate all costs on this sample
  - Configurable size, larger = more accurate
  - Derivation is like simulating a query
  - Even later this lecture
Important Note

- Derived estimations need not be exact
  - Should only help to discern good transformations from bad ones
  - Only order of alternatives matters, not their concrete cost
  - If 1st/2nd plan have estimated costs 1,1/1,15, although real costs are 10/1000, we nevertheless reach our goal – choosing the best

- Estimates in reality are often very bad
  - Orders of magnitude (see examples at end of this lecture)
  - 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
- Example: Oracle
- Some empirical observations
Rules of Thumb

- 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
  - $v(R, A)$: Number of **distinct values** of $A$
  - $\text{max}(R, A), \text{min}(R, A)$: Maximal/minimal value of $A$
    - Values that do exist in $R$, not maximal / minimal possible values
  - $|R|$: Number of tuples in $R$
  - Note: $R$ may be an **intermediate result**
Size after a Selection

• Assume \( \min \leq \text{const} \leq \max \)
• Selection of the \( S = \sigma_{A=\text{const}}(R) \)
  - \( |S| = |R| / v(R,A) \)
  - \( v(S,A) = 1; \max(S,A) = \min(S,A) = \text{const} \)
• 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. \( k=10,15,... \))
    • Idea: With such queries, one usually searches for outliers
    • \( k \sim \) frequency of outliers (“\text{magic constant}”)
    • Very rough estimate, but requires no knowledge of values in \( A \) at all
Selection II

• Selection of the form “A≠const”
  – |S| = |R| * (v(R,A)-1)/v(R,A)
    • We assume that const exists as value in A
  – v(S,A)=v(R,A)-1
  – min(S,A)=min, max(S,A)=max
  – Alternative model: |S| = |R|
Complex Selections

- **Conjunction**: Selection of the form \( A \theta c_1 \land B \theta c_2 \land \ldots \)
  - Assumption: Statistical independence of atomic conditions
  - Total selectivity is \( \text{sel}(c_1) \times \text{sel}(c_2) \times \ldots \)
  - \( \nu, \min, \max \) are adapted iteratively

- **Negation**: Selection of the form \( \neg A \theta c \)
  - Selectivity is \( 1 - \text{sel}(c) \)

- **Disjunction**: 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 \( 1 - (1 - \text{sel}(c_1)) \times (1 - \text{sel}(c_2)) \times \ldots ) \)

- Be careful: \( A < 55 \land A > 55 \)
Distinct and Projection

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

- Selectivity of projection
  - Projections usually only change the width of a tuple
    - Exception: Window functions
    - Width may increase: Computed attributes
  - Selectivity = 1 under **BAG semantics**

  - Caution
    - In real life, we need to estimate sizes in bytes
    - This requires **number of tuples and size of tuples**
    - Our current model ignores this issue
DISTINCT and GROUP-BY

• Selectivity of GROUP-BY
  – Same as selectivity of distinct on group attributes
• But: Selectivity of SELECT DISTINCT A,B,C FROM ...
Projection and Distinct

- **Selectivity of GROUP-BY**
  - Same as selectivity of distinct on group attributes
- **But: Selectivity of `SELECT DISTINCT A,B,C FROM ...`**
  - Not easy: We need to know correlations of values
  - Clearly, \( \leq < |S| \leq v(R,A) \times v(R,B) \times v(R,C) \)
  - Simple heuristic: \( |S| = \min\left( \frac{1}{2} |R|, v(R,A) \times v(R,B) \times v(R,C) \right) \)
- **Alternative**
  - Multi-dimensional histograms (later)
Selectivity of Cartesian Product

- Consider $S = R \times T$
  - $|S| = |R| \times |T|$
  - For all attributes $A$ of $S$: $\text{max}(S,A)$, $\text{min}(S,A)$, $v(S,A)$ are copied from base relation
Selectivity of Joins

- Consider join: $R \bowtie_A T$ (means $\sigma_{R.A=T.A}(R \times T)$)

- What is the 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 assume (or know!) joining 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
    - Then, 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 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)

• What about Theta-Joins: R⋈_{R.A<T.B} T ?
  - For each distinct value T.B, estimate which fraction of R has smaller values in R.A, then aggregate
Remarks

- We did not discuss effects of operations on other attributes
- Simple model: Ignore
  - Operation on R.A does not influence models of other attributes of R
  - Example: “age<19” does not change min(R,name) or max(R,name)
  - Often wrong: “age<19” does change max(R, income)
- In all other cases, we need to have models taking correlations of value sets into account
  - E.g. Multi-Dimensional Histograms
- As far as I know: Nowhere used in practice
Content of this Lecture

- Cost estimation
- Uniform distribution
- Histograms
- Sampling
- Example: Oracle
- Some empirical observations
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 optimization, **estimate selectivities** from affected buckets
    - Typical: Uniform distribution assumption inside each bucket
- **Advantage**
  - Hope: Frequencies **vary less inside smaller ranges**
  - **Lower errors** due to smaller ranges for uniformity assumption
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?
Distribution

- Assume normal distribution of weights
  - Spread: 120-40=80, mean: 80, stddev: 12; 100,000 people
- Uniform distribution: 100,000/80=1250 for each possible weight
- Leads to large errors in almost all possible query ranges
Equi-Width Histograms

- Fix number $b$ 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 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 single 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 \(\sim\) 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 \(\sim\) 11%
Example cont’d

• Query: Number of people with weight in [65-70]
  – Real value: 11603
  – Uniform distribution: \((70-65+1)*1250 = 7500\)
    • Error: 4103 \(\sim\) 35%
  – **Equi-depth** histogram
    • Range 65-69 has average 1850
    • Range 70-73 has average 2581
    • Estimation: \(5*1850 + 1*2581 = 11831\)
    • 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
  - Not directly applicable for REAL or VARCHAR (discretize!)
- 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 $O(b \times n^2)$ ($n$: |values|, $b$: |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>~72</td>
<td>~35</td>
</tr>
</tbody>
</table>
Other Types of Histograms

- **End-biased histograms**
  - Sort values by frequency and build singleton buckets for k 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
- Example: Oracle
- Some empirical observations
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!

```sql
SELECT count(*)
FROM sales S, reclamation R
WHERE S.productID=R.productID;
```

- 20K tuples
- 3K different values
- 380 tuples
- 250 different values
Example without Histograms

• Without histograms, assuming \textbf{uniform distribution}
  – Recall join-formula (no PK/FK)
  – Gives $|S| \times |R|/(\max (v(R,\text{productID}), v(S,\text{productID}))) \sim 2500$
Example with Histograms

- Uniform distribution within buckets
  - And uniform distribution of distinct values
    - Better: Store cnt of distinct value per bucket
    - \((7000 \times 300/500) + (450 \times 60/500) + ... \approx 4200\)
- More complicated if bucket borders of join attributes do not coincide
  - Always the case for equi-depth histograms

<table>
<thead>
<tr>
<th>Range</th>
<th>B.pID</th>
<th>R.pID</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>4900</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
• How to apply histograms for complex conditions?
  – People with weight<30 and age<25?
  – People with income>1M and tax depth<100K?
  – 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
Maintaining Equi-Width Histograms

- **Building**: Two scans
  - One for finding (min, max), one for counting bucket frequencies
    - Borders are regularly distributed over range
  - We can compute histograms for all attributes of a table at once

- **Maintaining**
  - If min / max does not change: Increase/ decrease frequencies in affected bucket
    - That’s the most frequent case: Maintaining **EW-histograms is cheap!**
    - Finding the bucket is in O(1)
  - If min/max does change: **Rebuild histogram**
    - Or ignore change and only change frequency in first/last bucket
Maintaining Equi-Depth Histograms

- **Building: Sort**
  - We need to sort all values, then partition into b roughly equal-size intervals
  - Requires one scan+sort per attribute
  - That’s rather expensive
    - Alternative: Use sample to estimate border values

- **Maintaining**
  - Almost all changes influence borders of buckets
    - Only updates of value within ED-range do not
  - Option 1: Accept intermediate inequalities in bucket frequencies
    - ... and regularly re-compute entire histogram
  - Option 2: Implement complex bucket merging/ splitting procedures
Offline Histograms

- Other option: Compute only on request and do not update
  - 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
- For long, this was the only available option
- Automatically maintaining statistics is an 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**
- Some empirical observations
Sampling

- Scanning a table for computing a histogram is expensive, yet accuracy is limited
  - If out-of-date, if conditions don’t match bucket borders, ...
- Other approach: Use a sample of the data
  - Reservoir sampling: Compute a random sample and maintain
  - If chosen randomly, sample should have same distribution as full data set – and also all correlations
  - Usually, a 1-5% sample suffices
    - The larger $|T|$, the smaller the percentage
- Also useful for approximate COUNT, AVG, SUM, etc.
  - Approximate query processing: Faster answers with small errors
  - Active research area (“Taming the terabyte”)
Building and Maintaining

- Idea: How to get a random sample of s% of table T?
  - Selecting first s% rows is a bad idea (yet fast)
  - Solution: Scan and pick every tuple with probability s
  - Will create a sample S of size roughly s*|T|
    - Exact size doesn’t matter
    - We just have to make sure that there is no buffer overflow

- Maintain
  - DELETE: If tuple in sample is deleted, choose new tuple at random
  - INSERT: Add new tuple to S with probability s
  - UPDATE: Propagate to sample
  - All this is expensive: Operations always need to check S
  - Alternative: Ignore and rebuild from time to time
Content of this Lecture

- Cost estimation
- Uniform distribution
- Histograms
- Sampling
- Example: Oracle
- Some empirical observations
Example: Oracle Basic Statistics

- Table statistics
  - Number of rows
  - Number of blocks
  - Average row length
- Column statistics
  - Number of distinct values (NDV) in column
  - Number of nulls in column
  - Data distribution (histogram)
- Index statistics
  - Number of leaf blocks
  - Levels
  - Clustering factor
- System statistics
  - I/O performance and utilization
  - CPU performance and utilization

- If activated: “Oracle gathers statistics on all database objects automatically and maintains those statistics in a regularly-scheduled maintenance job.”
- High-frequency tables: “Because the automatic statistics gathering runs during an overnight batch window, the statistics on tables which are significantly modified during the day may become stale”
Example: Oracle Histograms

- Most frequent values are very frequent
- Every bucket one value
- Equi-Depth

NDV > n

- Frequency Histogram
- Height-Balanced Histogram

ESTIMATE_PERCENT = AUTO_SAMPLE_SIZE

- No

Percentage of rows for top n frequent values ≥ p

- No

Top n Frequency Histogram

- Yes

\[ \text{NDV} = \text{Number of distinct values} \]
\[ n = \text{Number of histogram buckets (default is 254)} \]
\[ p = \left(1 - \frac{1}{n}\right) \times 100 \]
Content of this Lecture

- Cost estimation
- Uniform distribution
- Histograms
- Sampling
- Example: Oracle
- Some empirical observations
  - Leis, Gubichev, Mirchev, Boncz, Kemper, Neumann (2015): “How good are query optimizers, really?”, PVLDB
Empirical Observations

- **Goal**: Try to **separately measure** the relative impact of **cardinality estimation**, **cost model**, and **join order algorithm**
  - Hypothesis: Distribution assumptions (uniform) underlying most cost models are usually wrong
  - How much does this impact plan quality?

- **Approach**
  - **“Real-life” benchmark**: IMDB data, 21 tables, ~3GB raw data, many correlations between everything
    - Forget TPC-DS, TPC-H – synthetically generated (uniform) data
  - 33 query types with each ~3 incarnations; 113 queries, 3-16 joins
  - Use optimizers (with hints) to obtain cardinality estimates
  - Execute queries to obtain true cardinalities
  - Compare results from **five different database systems**
    - PostGreSQL, Hyper, DBMS-A, DBMS-B, DBMS-C
Selectivity of Selections on Base Tables

<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**

- 50% of estimates are almost perfect, in all systems
- 90% of estimates are wrong by a factor of 6 at most – but much worse in DBMS-C
- Extreme errors go up to factor 100.000
- Simple PostGres model works rather well
  - Min/max, distinct values, histograms
Cardinality Estimates for Multi-Joins

- All systems work rather well for up to 2 joins
  - With median errors below 10
- In all systems, accuracy decreases quickly with more joins
  - Note the logarithmic scale at y-axis
- Join sizes mostly are heavily underestimated
Do not Use TPC-H!

- Uniform data – perfect estimations
Impact on Runtime

- Approach: Obtain estimates from system X, inject into PostGres, let PostGres optimize and run the query
  - “Optimal”: Same approach using true cardinalities

<table>
<thead>
<tr>
<th></th>
<th>&lt;0.9</th>
<th>[0.9,1.1)</th>
<th>[1.1,2)</th>
<th>[2,10)</th>
<th>[10,100)</th>
<th>&gt;100</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL</td>
<td>1.8%</td>
<td>38%</td>
<td>25%</td>
<td>25%</td>
<td>5.3%</td>
<td>5.3%</td>
</tr>
<tr>
<td>DBMS A</td>
<td>2.7%</td>
<td>54%</td>
<td>21%</td>
<td>14%</td>
<td>0.9%</td>
<td>7.1%</td>
</tr>
<tr>
<td>DBMS B</td>
<td>0.9%</td>
<td>35%</td>
<td>18%</td>
<td>15%</td>
<td>7.1%</td>
<td>25%</td>
</tr>
<tr>
<td>DBMS C</td>
<td>1.8%</td>
<td>38%</td>
<td>35%</td>
<td>13%</td>
<td>7.1%</td>
<td>5.3%</td>
</tr>
<tr>
<td>HyPer</td>
<td>2.7%</td>
<td>37%</td>
<td>27%</td>
<td>19%</td>
<td>8.0%</td>
<td>6.2%</td>
</tr>
</tbody>
</table>

- Observations
  - Estimates from DBMS-A (HyPer) lead to near-optimal plans in 54% (37%) of all queries
  - DBMS-B (C, Hyper) estimates lead to plans more than 10 times slower than “optimal” for 32% (12%, 14%) of all queries
  - Overall: Even extremely bad estimates (DBMS-C) do not impact query performance too much too often
    - Wrong estimates sometimes even speed-up queries!
Quality of Cost Models

- Using **true cardinality** makes cost estimates much better
  - See different columns
- Changing the concrete cost model has little impact
  - See different rows
  - “Tuned”: MainMem-adapted
  - “Simple”: Roughly our option 1
- Message: **Invest in cardinality estimates**, not in performance modelling
But …

- More interesting results in the paper
  - E.g.: More indexes make estimations harder – larger search space
    - “Harder”, not “worse”

- But
  - A single data set
  - Real data, but synthetic workload
  - Runtimes are all from PostGres, ignoring many special features in the runtime engines of other systems
  - No parallelization
  - Although this data fits in memory, PostGres is not a MM-DBMS
    - Logs are writing to disk all the time

- Solution: Measure, model, and optimize for your workload