Module 9: Selectivity Estimation

Module Outline
9.1 Query Cost and Selectivity Estimation
9.2 Database profiles
9.3 Sampling
9.4 Statistics maintained by commercial DBMS

9.1 Query Cost and Selectivity Estimation

The DBMS has a number of alternative implementations available for each (logical) algebra operator.

Selecting an implementation for each of the operators in a plan is one of the tasks of the query optimizer.

The applicability of implementations may depend on various physical properties of the argument files (or streams), such as sort orders, availability of indexes, . . .

Among the applicable plans, the optimizer selects the ones with the least expected cost.

The cost of the implementation algorithms is determined by the size of the input arguments. Therefore, it is crucial to know, compute, or estimate the sizes of arguments.

This chapter presents some techniques to measure the quantitative characteristics of database files or input/output streams.

9.2 Database profiles

9.2.1 Simple profiles

Keep statistical information in the database catalogs. For each relation, record

- $|R|$ . . . the number of records for each relation $R$,
- $\|R\|$ . . . the number of disk pages allocated by those records,
- $s(R)$ . . . record size $s(R)$ and blocksize $b$ as an alternative ($\|R\| = \lfloor |R| / \lfloor b \rfloor \rfloor$)
- $V(A, R)$ . . . the number of distinct values of attribute $A$ in relation $R$,
- . . . and possibly more.

Based on these, develop a statistical model, typically, a very simple one, using primitive assumptions.

Principal Approaches

There are two principal approaches to estimate sizes of relations:

1. Maintain a database profile, i.e., statistical information about numbers and sizes of tuples, distribution of attribute values and the like, as part of the database catalog (meta information) during database updates.

   Calculate similar parameters for (intermediate) query results based upon a (simple) statistical model during query optimization.

   - Typically, the statistical model is based upon the uniformity and independence assumptions.
   - Both are typically not valid, but they allow for very simple calculations.
   - In order to provide more accuracy, the system can record histograms to more closely approximate value distributions.

2. Use sampling techniques, i.e., gather necessary characteristics of a query plan (input relations and intermediate results) by running the query on a small sample and by extrapolating to the full input size at query execution time.

   - Crucial decision here is to find the right balance between size of the sample and resulting accuracy of estimation.
Typical assumptions

In order to obtain simple formulae, assume one of the following:

▶ Uniformity & independence assumption: all values of an attribute appear with the same probability; values of different attributes are distributed independent of each other.

Simple, yet rarely realistic assumption.

▶ Worst case assumption: no knowledge available at all, in case of selections assume all records match the condition.

Unrealistic assumption, can only be used for computing upper bounds.

▶ Perfect knowledge assumption: exact distribution of values is known.

Unrealistic assumption, can only be used for computing lower bounds.

Typically use uniformity assumption.

9.2.1.1 Selectivity estimation under uniformity assumption

Using the parameters mentioned above, and assuming uniform and independent value distributions, we can compute the characteristics of intermediate results for the (logical) operators:

1. Selections \( Q := \sigma_{A = c}(R) \)

   Selectivity: \( \text{sel}(A = c) = 1/V(A, R) \) uniformity!

   Number records: \( |Q| = \text{sel}(A = c) \cdot |R| \)

   Record size: \( s(Q) = s(R) \)

   Number attribute values: \( V(A', Q) = \)
   
   \[
   \begin{cases} 
   1, & \text{for } A' = A, \\
   c(|R|, V(A, R), |Q|), & \text{otherwise.}
   \end{cases}
   \]

2. Other Selection Conditions

   ▶ Equality between attributes, e.g., \( Q := \sigma_{A = B}(R) \):
   
   An approximation for the selectivity could be
   
   \[
   \text{sel}(A = B) = \frac{1}{\max(V(A, R), V(B, R))} .
   \]

   This assumes that each value in the “smaller” attribute (i.e., the one with fewer distinct values) has a corresponding match in the other attribute.

   ▶ Range selections, e.g., \( Q := \sigma_{A > c}(R) \):
   
   If the system also keeps track of the minimum and maximum value of each attribute (denoted \( \text{High}(A, R) \) and \( \text{Low}(A, R) \) hereinafter), we could approximate the selectivity by
   
   \[
   \text{sel}(A > c) = \frac{\text{High}(A, R) - c}{\text{High}(A, R) - \text{Low}(A, R)} .
   \]

   ▶ Element tests, e.g., \( Q := \sigma_{A \in \{ \ldots \}}(R) \):
   
   An approximation for the selectivity could be obtained by multiplying the selectivity for an equality selection \( \text{sel}(A = c) \) with the number of elements in the list of values.
3. **Projections** $Q := \pi_L(R)$

Estimating the number of result tuples is difficult. Typically use:

$$|Q| = \begin{cases} 
V(A, R), & \text{for } L = \{A\}, \\
|R|, & \text{for key}(R) \in L, \\
\min(|R|, \prod_{A \in L} V(A, R)), & \text{without duplicate elimination},
\end{cases}$$

$$s(Q) = \sum_{A \in L} s(A)$$

$V(A, Q) = V(A, R)$ for $A \in L$

4. **Unions** $Q := R \cup S$

$$|Q| \leq |R| + |S|$$

$$s(Q) = s(R) + s(S)$$ same schema!

$$V(A, Q) \leq V(A, R) + V(A, S)$$

5. **Differences** $Q := R - S$

$$\max(0, |R| - |S|) \leq |Q| \leq |R|$$

$$s(Q) = s(R) = s(S)$$ same schema!

$$V(A, Q) \leq V(A, R)$$

6. **Products** $Q := R \times S$

$$|Q| = |R| \cdot |S|$$

$$s(Q) = s(R) + s(S)$$

$$V(A, Q) = \begin{cases} 
V(A, R), & \text{for } A \in \text{sch}(R) \\
V(A, S), & \text{for } A \in \text{sch}(S)
\end{cases}$$

7. **Joins** $Q := R \bowtie_F S$

This is the most challenging operator for selectivity estimation!

- A few simple cases are:
  - No common attributes ($\text{sch}(R) \cap \text{sch}(S) = \emptyset$), or join predicate “$F = \text{true}$”:
    $$R \bowtie_F S = R \times S$$
  - Join attribute, say $A$, is key in one of the relations, e.g., in $R$, and assuming the inclusion dependency “$\pi_A(S) \subseteq \pi_A(R)$”:
    $$|Q| = |R|$$

- In the more general case, again assuming inclusion dependencies, “$\pi_A(S) \subseteq \pi_A(R)$” or “$\pi_A(R) \subseteq \pi_A(S)$”, we can use two estimates:

$$|Q| = \frac{|R| \cdot |S|}{V(A, R)}$$ or $$|Q| = \frac{|R| \cdot |S|}{V(A, S)}$$

typically use the smaller of those two estimates, i.e.,

$$|Q| = \frac{|R| \cdot |S|}{\max(V(A, R), V(A, S))}$$

$$s(Q) = s(R) + s(S) - \sum_{A \in \text{sch}(R) \cap \text{sch}(S)} s(A)$$ for natural join

$$V(A', Q) \leq \begin{cases} 
\min(V(A', R), V(A', S)), & \text{for } A' \in \text{sch}(R) \cap \text{sch}(S) \\
V(A', X), & \text{for } A' \in \text{sch}(X)
\end{cases}$$
9.2.1.2 Selectivity estimation for composite predicates

For selections with composite predicates, we compute the selectivities of the individual parts of the condition and combine them appropriately.

Combining estimates ...?

Here we need our second (unrealistic but simplifying) assumption: we assume independence of attribute value distributions, for under this assumption, we can easily compute:

- Conjunctive predicates, e.g., \( Q := \sigma_{A = c_1, B = c_2}(R) \):
  \[
  \text{sel}(A = c_1 \land B = c_2) = \text{sel}(A = c_1) \cdot \text{sel}(B = c_2),
  \]
  which gives \(|Q| = \frac{|R|}{V(A, R) \cdot V(B, R)}\).

- Disjunctive predicates, e.g., \( Q := \sigma_{A = c_1 \lor B = c_2}(R) \):
  \[
  \text{sel}(A = c_1 \lor B = c_2) = \text{sel}(A = c_1) + \text{sel}(B = c_2) - \text{sel}(A = c_1) \cdot \text{sel}(B = c_2),
  \]
  which gives \(|Q| = \frac{|R|}{V(A, R) + V(B, R) - V(A, R) \cdot V(B, R)}\).

9.2.2 Histograms

Observation: in most cases, attribute values are not uniformly distributed across the domain of an attribute.

We need to keep track of non-uniform value distribution of an attribute \( A \), maintain a histogram to approximate the actual distribution:

1. Divide attribute domain into adjacent intervals by selecting boundaries \( b_i \in \text{dom}(A) \).
2. Collect statistical parameters for each such interval, e.g.,
   - number of tuples with \( b_{i-1} < t(A) < b_i \),
   - number of distinct \( A \)-values in that interval.

Two types of histograms can be used:

- "equi-width histograms": intervals all have the same length,
- "equi-depth histograms": intervals all contain the same number of tuples

Histograms allow for more exact estimates of (equality and range) selections.

Example of Approximations Using Histograms

For skewed, i.e., non-uniform, data distributions, working without histograms gives bad estimates. For example, we would compute a selectivity \( \text{sel}(A > 13) = \frac{48}{10} = 3 \), which is far from exact, since we see that, in fact, 9 tuples qualify.

The error in estimation using the uniformity assumption is especially large for those values, which occur very often in the database. This is bad, since for these, we would actually need the best estimates!

Histograms partition the range of (actual) values into smaller pieces ("buckets") and keep the number of tuples and/or distinct values for each of those intervals.

Example of Approximations Using Histograms (cont’d)

With the equi-width histogram given below, we would estimate 5 result tuples, since the selection covers a third of the last "bucket" of the histogram (assuming uniform distribution within each "bucket" of the histogram).

Using the equi-depth histogram also given below, we would, in this case, estimate the exact result (9 tuples), since the corresponding "bucket" contains only a single attribute value.

Typically, equi-depth histograms provide better estimates than equi-width histograms. "Compressed histograms" keep separate counts for the most frequent values (say, 7 and 14 in our example) and maintain an equi-depth (or whatever) histogram of the remaining values.
9.3 Sampling

**Idea:** if maintaining a database profile is costly and error-prone (i.e., may provide far-off estimates), may be it is better to dispense with this idea at all. Rather, if a complex query is to be optimized, execute the query on a small sample to collect accurate statistics.

**Problem:**
- How “small” shall/can the sample be?
  - Small enough to be executed efficiently, but large enough to obtain useful characteristics.
- How precisely can we extrapolate?
  - Good prediction requires large (enough) samples.

... select a value for either one of those parameters and derive the other one from that.

9.4 Statistics maintained by commercial DBMS

The research prototype System/R introduced the concept of a database profile recording simple statistical parameters about
- each stored relation,
- each defined index,
- each segment (“table space”).

In addition to the parameters mentioned above (tuple and page cardinalities, number of distinct values), the system stores the current minimum and maximum attribute values (to keep track of the active domain and to provide input for the estimation of range queries). For each index (only B+ tree-indexes are supported by System/R), store height of the index tree and number of leaf nodes.

Many commercial DBMSs have adopted the same or similar catalog information for their optimizers.

### Three approaches found in the literature

1. “adaptive sampling”: try to achieve a given precision with minimal sample size.
2. “double (two-phase) sampling”: first, obtain a coarse picture from a very small sample, just good enough to calculate necessary sample size for a “useful” precision in second step.
3. “sequential sampling”: sliding calculation of characteristics, stops when estimated precision is good enough.

### Parameters of a DB profile in the System/R catalogs

| per segment | \(NP\) | number of used pages |
| per relation | \(\text{Card}(R)\) | \(|R|\), number of tuples |
| | \(\text{PCard}(R)\) | \(\|R\|\), number of blocks |
| per index | \(\text{ICard}(I)\) | \(V(A, R)\) ... number of distinct values in indexed attribute |
| | \(\text{MinKey}(I)\) | minimum attribute value in indexed attribute |
| | \(\text{MaxKey}(I)\) | maximum attribute value in indexed attribute |
| | \(\text{NIndx}(I)\) | number of leaf nodes |
| | \(\text{NLevels}(I)\) | height of index tree |
| | \(\text{Clustered}?)\) | is this a clustered index (yes/no) |

All values are approximations only! They are not maintained during database updates (to avoid “hot spots”), rather they can be updated explicitly via SQL’s “update statistics” command.

**N.B.**: per segment information is used for estimating the cost of a segment scan. System/R stores more than one relation in a segment and uses the TID addressing scheme. Therefore, there is no way of doing a relation scan. If no other plan is available, the system will have to scan all pages in a segment!
System/R estimation of selectivities

Obviously, System/R can use $I\text{Card}(I)$-values for estimating the selectivity $\text{sel}(A = c)$ for simple “attribute-equals-constant”-selections in those cases, where an index on attribute $A$ is available.

But, what can we do for the other cases?

$\triangleright$ If there is no index on $A$, System/R arbitrarily assumes $\text{sel}(A = c) = \frac{1}{10}$.

$\triangleright$ For selection conditions $A = B$, the system uses

$$\text{sel}(A = B) = \frac{1}{\max(\text{Card}(l_1), \text{Card}(l_2))},$$

if indexes $l_1$ and $l_2$ are available for attributes $A$ and $B$, respectively.

This estimation assumes an inclusion dependency, i.e., each value from the smaller index, say $l_1$, has a matching value in the other index. Then, given a value $a$ for $A$, assume that each of the $\text{ICard}(l_2)$ values for $B$ is equally likely. Hence, the number of tuples that have a given $A$-value $a$ as their $B$-value is $\frac{1}{\text{Card}(l_2)}$.

If only one attribute has an index, assume selectivity $\text{sel}(A = B) = \frac{1}{\text{Card}(l_2)}$; if neither attribute has an index, assume the ubiquitous $\frac{1}{10}$.

Other systems

$\triangleright$ DB2, Informix, and Oracle use one-dimensional equal height histograms. Oracle switches to duplicate counts for each value, whenever there are only few distinct values.

$\triangleright$ MS SQL Server uses one-dimensional equal area histograms with some optimization (compression of adjacent ranges with similar distributions). SQL Server creates and maintains histograms automatically, without user interaction.

$\triangleright$ Sampling is typically not used directly in commercial systems. Sometimes, utilities use sampling for estimating statistics or for building histograms. Sometimes sampling is used for load balancing during parallelization.

$\triangleright$ These formulae are used whether or not $A$ and $B$ are from the same relation. Notice the correspondence with our estimation of join selectivity above!

... System/R estimation of selectivities (cont’d)

$\triangleright$ For range selections $A > c$, exploit the $\text{MinKey}$ and $\text{MaxKey}$ parameters, if those are present (i.e., if an index is available).

$$\text{sel}(A > c) = \frac{\text{MaxKey}(I) - c}{\text{MaxKey}(I) - \text{MinKey}(I)}$$

If $A$ is not an arithmetic type or there is no index, a fraction less than half is arbitrarily chosen. Similar estimates can be derived for other forms of range selections.

$\triangleright$ For selections of the form $A \in \text{IN} \ (\text{List of values})$, compute the selectivity of $A = c$ and multiply by the number of values in the list. Note that this number can be the result of a complex selectivity estimation in case of SQL’s nested subqueries.

However, never use a resulting value greater than $\frac{1}{2}$, since we “believe” that each selection eliminates at least half of the input tuples.

Bibliography


