Query Processing: Cost-based Query Optimization
Join Operation

Estimating the size of join

For $R \bowtie_C S$ operation:

- Consider the cost as the cost of $\sigma_C(R \times S)$. We know that $T(R \times S) = T(R) \cdot T(S)$, so, we can now apply our estimates for selection operation to it. (Estimate selectivity of an inequality comparison, e.g., $R.A < S.B$ as $\frac{1}{3}$).

For $R \bowtie S$:

- If R and S have disjoint values: $T(R \bowtie S) = 0$.
- If R and S are joined on a one-to-one foreign key, then $T(R \bowtie S) = \min(T(R), T(S))$.
- If R and S are joined on a many-to-one foreign key, then $T(R \bowtie S) = \max(T(R), T(S))$ (actually, it is the size of the table that contains the foreign key, which would *typically* be the larger table).
- If R and S have many tuples to be joined (join attributes have uniformly many values), then $T(R \bowtie S) = O(T(R) \cdot T(S))$.

To develop better estimates, we consider two asspumptions:

1. Containment of value sets. Assume that if Y is an attribute in both R and S, then if $V(R, Y) \geq V(S, Y)$ then all Y-values in S are also found in R.
2. Preservation of value sets. IfA is an attribute in R but not in S, then $V(R, A) = V(R \bowtie S, A)$.

Under these assumptions:

- For natural joins/equijoins *with a single join attribute*:

$$T(R \bowtie S) = \frac{T(R) \cdot T(S)}{\max(V(R, Y), V(S, Y))}$$

- For natural joins/equijoins *with multiple join attributes*. Let $A_1, \ldots, A_k$ be the join attributes:

$$T(R \bowtie S) = \frac{T(R) \cdot T(S)}{\Pi_{i=1}^{k}(\max(V(R, A_i), V(S, A_i)))}$$

Computing $B(R \bowtie S)$. Note that $R \bowtie S$ has a different schema than either R or S (unless both R and S have the same schema, and then, $R \bowtie S = R \cap S$). Usually, a record from $R \bowtie S$ will be larger than a record from either R or S. Thus, in addition to computing $T(R \bowtie S)$, we need to compute $B(R \bowtie S)$.

The computation is similar to the computation of $B(\pi_L(R))$. Let m_1 be size of a record from r, m_2 – the size of a record from S, and m be their common part. Then estimate

$$B(R \bowtie S) = \frac{T(R \bowtie S)}{B} \cdot (m_1 + m_2 - m)$$

Selecting Join Order

Some join algorithms (hash-based, nested loops) treat relations assymetrically.

- Build relation: the (usually smaller) relation that is put in main memory;
- Probe relation: the relation that is scanned;

We assume that in expression $R \bowtie S$, $B(R) \leq B(S)$, and R is the build relation, while S is the probe relation.

Join of two relations. Select the smaller relation as the build, select the larger as the `probe`.

Join of three or more relations. There are multiple tree shapes that are possible for the logical query plan for a join of three or more tables. Typically, only the left-deep join trees are considered. A binary tree is left-deep if all right children of all its nodes are *leaves*.

Use of left-deep join trees reduces the problem of selecting a plan for join to the problem of ordering tables. Still, for n tables in the join, there are $n!$ ways to order them.

- Exhaustive enumeration. Too expensive!
- Look at a subset of orderings. *dynamic programming*.
- Use a heuristic. *greedy algorithm*.