Summary of ”How Good Are Query Optimizers, Really?”

By S. Xiao Fernández Marín

1 Summary

The problem, of finding a good join order is one of the most studied problems in the DB field as it is crucial for performance. This paper answers some research questions: *How good are cardinality estimates and when do bad estimates lead to slow queries?*, *How important is an accurate cost model? (The cost formulas in the optimizer)*, *How large does the enumerated plan space need to be? (Enumerating all plans carries a cost)*.

One of the main contributors of this paper is the design of the Join Order Benchmark (JOB). Based on the IMDB data set of 3.6GB and a total of 113 queries. It is focused on joins, join order, join method, access method and no ”group by” or advanced SQL. It was inspired by the benchmarks **TPC-H**, **TPC-DS** and **SSB**. They did not correlate and they were not good at testing cardinality estimates. They also assume uniformly distributed data and the cardinality estimation was harder and was based on unrealistic assumptions: Uniformity, Independence and Principle of inclusion.

Having the JOB benchmark, they did dome experiments, a setup of 5 different DBMS: PostgresSQL, HyPer and 3 other unnamed commercial DBMS. **Postgres** was the main testbench, they modified it to make it possible to inject cardinality estimates and in some experiments, they modified the cost model for the main memory setting. In **HyPer** the main memory DBMS was from the same research group.

The cardinality estimation is the most important thing for a good query plan. That is why we need it to be with the most quality we can achieve. The selection is made from a single table and may contain complex predicates to measure the quality of base tables cardinality estimates, we use the **q-error**, which is the factor between the estimate differs and the actual cardinality.

The next table shows how the percentiles of q-error of 629 base tables:

	median	**90th**	**95th**	**max**
PostgreSQL	1.00	2.08	6.10	207
DBMS A	1.01	1.33	1.98	43.4
DBMS B	1.00	6.03	30.2	104000
DBMS C	1.06	1677	5367	20471
HyPer	1.02	4.47	8.00	2084

Table 1: Q-errors for base table selections

From Table 1, we can deduce that it is based on per-column histograms and we also see that HyPer and DBMS A use samplings instead of histograms, as they have better estimates. There is also displayed a boxplot where the estimation of intermediate results for joins is shown. As a result, DBSM B gradually gets worse. Hyper, DBMS C and PostgreSQL are less the same and DBMS A is the best one, assuming it has a damping factor that depends on the join size.

We are also shown the estimates for 3 TPC-H queries vs 4 JOB queries (clearly being the TPC-H the one that presents many hard challenges) and PostgreSQL based on default and true distinct counts, confirming that the true distinct counts slightly improves the variance of errors but the trend to underestimate cardinalities becomes even more pronounced.

After studying this, we can investigate the conditions under which bad cardinalities are likely to cause slow queries. To measure the impact of cardinality misestimation on query performance we injected the estimates of the different systems into PostgreSQL and then executed the resulting plans. We get a table like the next one:

Where a small number of queries become slightly slower using the truth instead of the erroneous

	<0.9	[0.9,1.1)	[1.1,2)	[2,10)	[10,100)	>100
PostgreSQL	1.8%	38%	25%	25%	5.3%	5.3%
DBMS A	2.7%	54%	21%	14%	0.9%	7.1%
DBMS B	0.9%	35%	18%	15%	7.1%	25%
DBMS C	1.8%	38%	35%	13%	7.1%	5.3%
HyPer	2.7%	37%	27%	19%	8.0%	6.2%

Table 2: Runtimes based on slow down wrt

cardinalities.

We are also shown a plot where it is shown the slowdown of queries using PostgreSQL estimates wrt by default, with no nested-loop join and with rehashing, where we can confirm that being "purely cost-based" can lead to bad query plans. There is also a plot of how slowdown of queries using PostgreSQL estimates wrt using true cardinalities.

The cost model is also a factor to take into account where we should find a linear relationship between the predicted cost and the runtime for different cost models.

The planned space is an algorithm that explores the space of semantically equivalent join orders so it can choose the best plan. We will see how large this needs to be.

Plan space also looks if bushy trees are necessary. We have left and right deep trees, and the zig-zag tree. The optimal plan and the best plan are compared. We get that in the former, all the methods are good and in the latter, the zig-zag is the best and the right-deep the worst.

Now, they are also comparing greedy heuristics with the dynamic programming and quick pick-1000, taking into account that they are using the optimal plan and the best plan are compared which in conclusion, if you have PostgreSQL in the latter, it will be the worst option while if you have true cardinalities in the optimal plan or best plan, the dynamic programming is always going to be the best option.

In conclusion, The second route would be to increase the interaction between the runtime and the query optimizer. We leave the evaluation of both approaches for future work.

2 Questions not answered by this text

DBMS can have more advanced estimation algorithms like Multidimensional Partitioning or Mapping Multidimensional Data

Another way of improving a plan quality in heavily indexed settings would be to increase the interaction between the runtime and the query optimizer.

3 What has changed since this was written

Nowadays, the JOB is not enough for evaluating cardinality estimation, so there are developing new technologies using machine learning like the cardinality estimation benchmark.