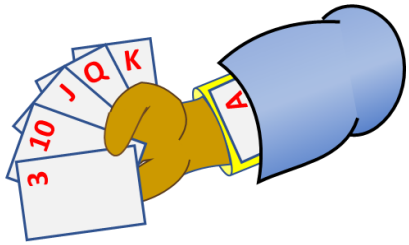
The Masked Analyst Reveals Benchmark's Biggest Tricks and Cheats

By [redacted], Benchmark Trainer and Mentor

In a television show shown in the UK, Canada, Australia, and the US in 1997-1999, a Brazilian born magician, Val Valentino, donned a mask and starred in a show where he broke the Magician's Code not to reveal the secrets of their tricks. Several lawsuits followed as some magicians had spent 10s of thousands of dollars on the equipment for their tricks and others had developed tricks that they had licensed to other magicians.



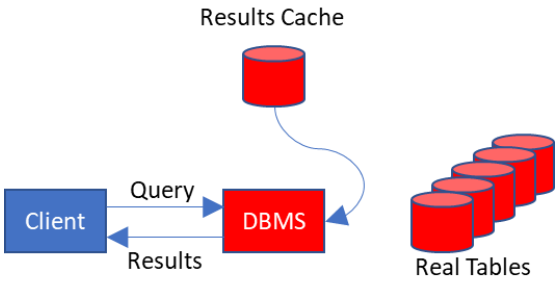
In this era of Covid … and wearing a mask, I have chosen to break the benchmark analyst's unwritten code of silence to reveal the secrets that DBMS vendors use to win business.

Val believed disclosing the secrets behind tricks would encourage others to become magicians and shake up the magical arts as magicians had become complacent. I doubt that revealing benchmark secrets will encourage anyone to become benchmark analysts unless they are predisposed to the insanity of building a data warehouse in a few weeks with incomplete table definitions, malformed source data, queries that never worked and a deadline that never changes but a start date that is repeatedly delayed. It is my hope that revealing these secrets will shake up benchmark practices and result in more realistic benchmarks that help companies choose the best technology to meet their production needs.

Trick #1: Results Caching

Some DBMSs can save the final spool file of results along with information about the query that created it (like the hash value of the source code). If a matching query is executed, the DBMS merely returns the result set from the prior execution.

In a benchmark run by a BI tool vendor, they allowed results caching to stay on. They argued, in normal production usage there would be a repeat of the execution of some common queries. The problem is, in a benchmark there is not the robust variety of queries found in production. In their 106-query test, they ran a "warm-up" test before the measured test which had the effect of populating the results cache. We created a RedShift DBMS and re-ran their tests with and without results caching. The result:



- **Without results cache:** 106 queries took 125.33 seconds and used 1,692 seconds of CPU
- **With results cache**: 106 queries took 7.74 seconds and used 179 seconds of CPU. Only 2 queries were re-executed because they produced larger result sets

All DBMSs we are aware of that use results caching have statements to turn it off. Attempts to defeat results caching by parameterizing queries may have inconsistent success based on the amount of storage available for results caching and the logic DBMSs use for retaining results. We've seen sets of results retained for as long as 24 hours. They may even be retained across DBMS restarts.

Trick #2: Select Count(*)

A common problem in running queries in a benchmark is that they may return a huge number of rows. Much of the execution time of a query could be the returning of rows in which case the network speed and network latency would play a large role in the perceived response time. To address this, some analysts will wrap the business queries with "select count(*) from (… *business query* …);"

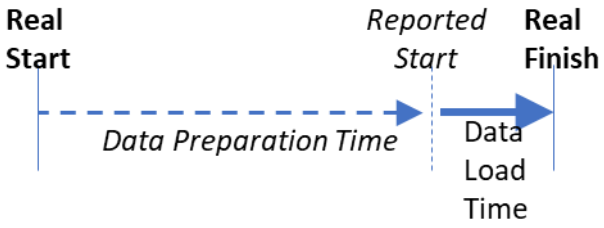
> Select count(*) from (
> Select count distinct
> people in the U.S.A.)
> … the answer … hmmm … 1?

The problem is that the optimizer may realize that it doesn't need to produce the rows in spool to answer the question: "How many rows would be produced?" Consider a complex 20 table join with statistical functions like RANK and complex IN lists and constraints with the purpose of finding the total revenue of the top 10 customers. If there is no "GROUP BY" clause, the optimizer will recognize that the answer of the "SELECT COUNT(*)" will be 1, and the seemingly complex query will complete in 0.01 seconds. Even with a "GROUP BY" clause, it may not be necessary to SUM or RANK to say how many rows will be produced, so only the keys may be processed through spool files.

One competitive benchmark consisted of 8 queries. When those queries ran normally, they consumed 402,813 CPU seconds and produced an average of 199,212,977 rows of output. The analyst wrapped the queries with "select count(*) from (…)" to avoid the millions of output lines that each produced. With the "select count()" wrapper, the 8 queries only used 15 seconds of CPU, **saving 99.9963% of the processing**.

Trick #3: Prestaging Data

In one benchmark, the DBMS vendor was faced with loading 23.8 TB to 227 tables. They requested that the customer break up the data loading files into 10 GB chunks. As a result, there were over 80,000 files. That vendor impressed the customer with their data loading performance, taking less that 24 hours to load the 23.8 TB. Unfortunately, not counted was the time that was spent splitting up the files into small enough chunks for the data load utility to deliver good performance.



Trick #4: Pre-Sorting Data

Some DBMSs record the beginning and ending values of some column in a storage unit, (such as a data block or group of blocks), in a zone map. This enables the DMBS to prune the data blocks needed to be processed when a query uses a constraint on the column that is the basis of the map. By presorting the data, it is more quickly loaded into the zone map. ***Note**: This is not an argument against using DBMS features that improve performance. However, like Trick #3, include the sorting in the preparation time.*

Some DBMSs use vertical compression where the only data stored is that data which is different from the prior row. Depending on the amount of denormalization in the data, this can result in impressive space savings and reduced I/O when the presorted data is scanned.

Order_dt	Cust_no	Inv_no	Prod_cd	Qty	Cost
2020/02/11	213896	5311444	12133	2	10.50
2020/02/11	213896	5311444	18712	1	2.40
2020/02/11	213896	5311444	21321	1	1.50
2020/02/11	213896	5311444	22101	1	1.50

Unfortunately, this pre-sorting of data works better in the preparation of the initial data load of a benchmark than it does in real life where transactions and dimension table changes could happen every day, hour or even minute-by-minute. For Teradata, performance may be worse if the data is sorted by the primary index because Teradata merges data into tables by the hash value of the primary index and sorting may cause transient skewed performance.

Trick #5: Not Reporting Failed Queries

A benchmark may consist of hundreds or thousands of queries. (A BI report may be created by the BI tool using temporary tables, processing 2 at a time, and use over 100 queries for a single report). It is not unusual to have some number of queries fail on the first execution, either because of missing database objects or differences in the functions or syntax supported by the DBMS.

Failing to resolve those failing queries will yield a lower total run time.

	RetCode	Seconds	Rows
Qry01	0	21.33	45
Qry02	8130	.01	0
Qry03	0	12.72	314
Qry04	0	2.96	0
Qry05	0	8.74	89
Total for 5? Queries:		45.76	

Trick #6: Not Validating Row Counts

It is not unusual when a snapshot of production data is exported for the benchmark on one day and the queries are extracted on a different day, that the exported data may not have the date ranges or transaction keys that were present when the query ran. The query may have been converted from another DBMS and an error in the revised logic does not produce the same results as the original query. The revision may have simplified the processing, yielding an erroneous performance advantage.

It is important to at least compare the row counts from one execution of each query with the production execution to ensure that there are no gross errors in the conversion of the query from a different DBMS or its relation to the time period of the data snapshot.

Trick #7: Not Loading All Data

In the exporting of data, mistakes are often made, such as exporting text fields containing unprotected delimiters or line ends. The result could be a rejection of some number of rows due to the formatting error. On the other hand, there could be columns used in joins that have a lot of duplicate values that

Snapshot			Production	
Tbl1	45	=	Tbl1	45
Tbl2	10,131	=	Tbl2	10,131
Tbl3	41,501	??	Tbl3	41,491
Tbl4	31,989	=	Tbl4	31,989

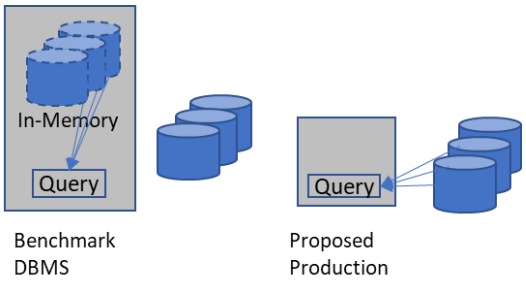
result in an unusual amount of processing that can be eliminated by dropping a couple of rows from a dimension table, eliminating the processing issue.

Some tables could have been defined in the DDL, but the snapshot of data failed to include that table, or the vendor failed to load it. Running queries against empty tables produces really fast execution time, but is not representative of what your selected system will need to do.

It is best practice to compare the row counts between the exported snapshot and the data loaded for the benchmark.

Trick #8: Oversizing Benchmark Platform

Vendors may attempt to win the speed contest by using a larger platform during the contest than the customer will likely purchase. They will argue that the smaller, less expensive system will have proportional speed. In fact, the larger platform may be able to hold much of the data in memory and avoid disk I/O.



Best practice is to disclose the platforms being used by each vendor in a contest to all vendors so they can challenge the competitor's sizing for the benchmark.

Trick #9: Pinning Tables in Cache or Memory

Related to #8 is the practice of tuning for the benchmark tests by pinning some tables in memory. This may not be practical in a production environment with a variety of queries and SLAs, but for the benchmark, reducing or eliminating physical I/Os is a great way to improve the chances of winning.

You need to engage experts in the platform or review the vendor documentation to see if they support this and then consider if using that in the benchmark represents something you are likely to do in production.

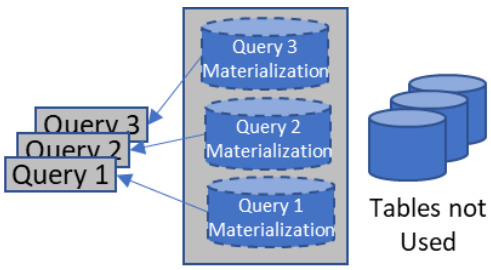
Trick #10: Query Specific Hints

Some optimizers can accept hints to override the query plan that would be automatically created. While this may be practical in a benchmark with 20 to 200 queries, is that something you could sustain in a production environment with hundreds of jobs and thousands of ad hoc users?

Best practice is to examine the queries exactly as they've been run in the benchmark and ask yourself if the things required to get good performance are sustainable in a production environment.

Trick #11: Query Specific Materializations

View materializations and aggregate join indexes are powerful DBMS tools to improve response time. However, there may be a tendency to resolve performance problems on specific queries by creating a materialization or a join index for the poorly performing query.



This is not a suggestion that you should prevent vendors from using these tools to improve workload performance,

but when they only apply to a single query, you should challenge whether that approach would be practical in production. Best practice is to get all DDL involved in a benchmark and where materializations or join indexes are used, analyze whether they are generally applicable or specific to one query.

Trick #12: Running Benchmarks Unsupervised

There is some work in a benchmark that is mundane as the vendor figures out what tables need to be created, loaded and queries adapted. Sometimes customers insist on being "in the room" for every step in the preparation. This is overkill.

However, it is a good practice to check in on the work as the benchmark is being prepared. It is essential to be involved as the benchmark is being executed to detect any shenanigans, to get firsthand experience at how the platform performs, and to observe any potential support requirements.

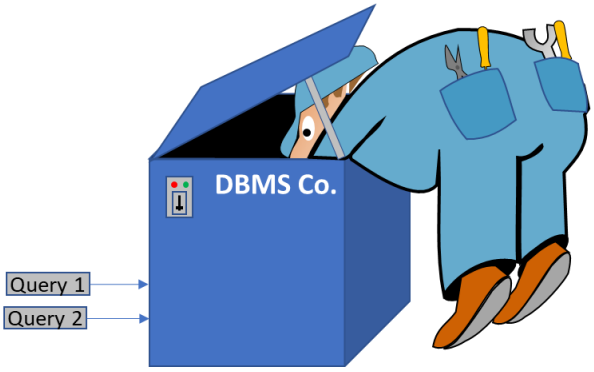
If the vendor insists on running the benchmark at their site, insist on being present. Don't be distracted by presentations and opportunities to be wined and dined.

Use change dates on files and objects to detect what changes were made while you weren't observing the work. Insist that changes to queries have the original line commented out and a comment on the reason for the change ahead of the new line(s) of code.

Trick #13: Proposing Simplistic Benchmarks

Some vendors have been known to say, "give me your worst 5 queries and we will prove we are faster than your current platform." They propose that for two reasons:

1. It is cheap for them to execute
2. It is much easier to tune the DBMS



With only a handful of queries, they can intently focus on making those queries perform well. They may run them over and over allowing data to be cached to improve performance.

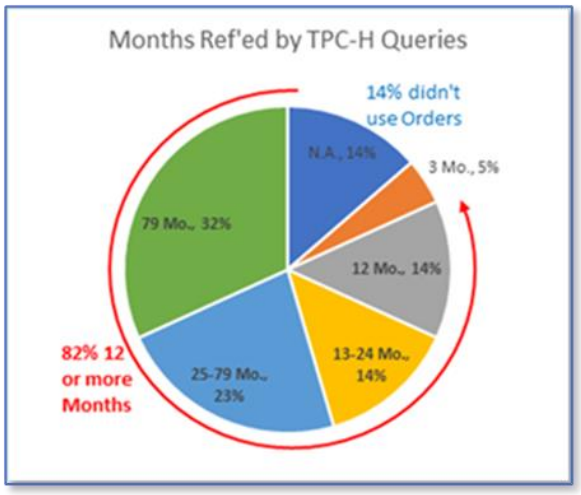
The problem is, this is not representative of the production workload that the selected platform will need to support. The queries selected may hit that vendor's "sweet spot" and while they may perform better with those queries, they may fail with the mix of queries that represent your actual workload. Make sure you test with queries that are representative of your actual production needs.

Trick #14: Running A Benchmark with an "Industry Standard" Benchmark

Some vendors will propose running "Industry Standard Benchmarks" … *(pause for the horn fanfare, then say the following with emphasis)* .. because they are **industry standard** and **free** from **any vendor bias**. … oh really? Then why are they eager to use them?

TPC-H for example, was released in April, 1999 consisting of 22 queries against 8 tables. The queries are heavy scan oriented and tend to reference a large amount of data. In profiling data warehouses across industries, we find that a large percentage of queries support tactical needs of the business such:

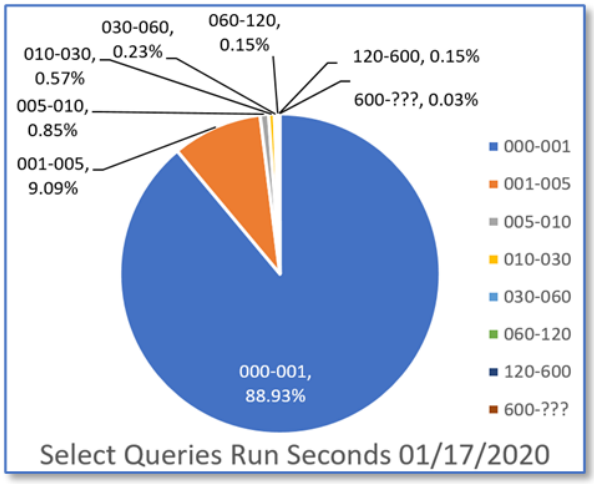
- as an order clerk checking on an order,
- or a credit analyst looking at the transactions of a customer behind on payments,
- or a sales representative preparing to visit a customer,
- or a brand manager tracking new item sales or the impact of a new promotion.



However, in TPC-H, 82% of the queries scan 12 or more months of order data.

The typical data warehouse profile has 60-90% of the queries executing in 1 second or less. Why don't vendors propose to benchmark that type of workload? Could it be:

- Because they can't compete with that volume of tactical queries?
- Or, maybe they've had 20 years of tuning experience with the **Industry Standard Benchmarks?**



Best practice is to profile your current production workload and then select a representative workload with a similar profile in terms of statement type, run seconds or I/O, and number of tables or steps executed by the query.

Benchmark tests should be run at various concurrencies using a query driver that can execute queries in the same ratio as your intended production so you can feel confident your selected DBMS will meet your needs.

Closing Thoughts

You may be looking for a vendor that can partner with you to provide technology that can advance the analytics supporting your business processes. I hope this document doesn't motivate you to rash actions like trying to execute the benchmark without vendor resources. They have the expertise and your DBAs have day jobs, so you need to rely on the vendors.

My advice? Ronald Regan, 40th president of the US, was taught Russian in preparation for nuclear disarmament talks with Russian president, Mikhail Gorbachev. He picked up a Russian proverb: "Доверяй, но проверяй" which in English (without the rhyme) is: **Trust but verify.**