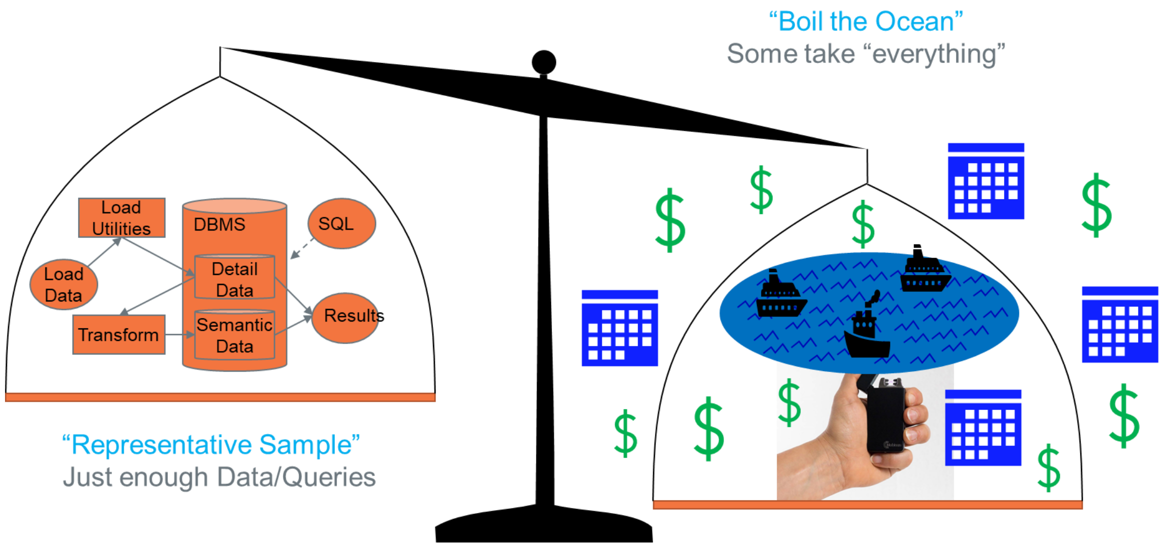
Benchmark Deception and How to Avoid It

By Douglas H Ebel, Benchmark Trainer and Mentor

You are probably reading this document because you want to know how to design a benchmark that avoids the issues outlined in the document "Benchmark Deception". That document outlined how many benchmarks do not prove that the selected DBMS will meet a company's production needs. This is a guide to selecting a benchmark workload that is representative of the production requirements without "boiling the ocean" and doing a complete conversion that is costly and delays a decision. It is a balancing act:



The key is to success is:

1. Analyze your current production workload profile
2. Select a sample workload
3. Trim and replicate that sample workload to have a profile like production
4. Adapt queries for use in a benchmark
5. Snapshot the production data, protecting sensitive and PII data
6. Select the mechanism and model for executing the benchmark
7. Design the benchmark tests with various concurrencies and data maintenance activities to simulate the demands that will be placed on the selected DBMS

Step 1: Understand Your Production Needs

Begin by profiling your current workload. Recognizing that no day has the same processing requirements of every other day, it is sufficient to pick a representative day of processing for your analysis. It is best to choose a relatively heavy processing day but not one with unusual events like recovery after a system down.

Your analysis should break down queries by run seconds, number of tables, application, steps and summarize the CPU and IO. Your benchmark workload should have similar ratios but be much smaller.

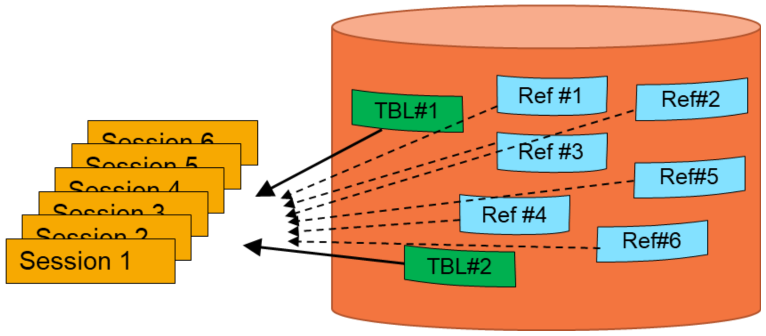
You should also query each minute to determine how many queries were in flight and how many were waiting to run. Too often, it is assumed that if there are 1,000 users that it is necessary to execute 1,000 queries concurrently. That could only happen if after the annual meeting where the executive gave an inspiring pep talk, that everyone raced back to their desk to compose a query and press the Enter key all at once. The reality is that with 1,000 potential users, a heavy period would see 100 users logged on. With the time to compose queries, review query results, take phone calls, get coffee, etc., it is likely that those 100 users would have 10 queries in flight.

The 10:1 rule

1,000 users =
100 logged on =
10 queries in flight

Step 2: Select a Sample Workload

Designate several key tables that are important to your business. Determine which sessions referenced those tables and which other related tables those sessions would need. It is best to do this analysis by session instead of by query since Business Intelligence queries and power users may create a report using a sequence of queries with intermediate temporary tables.



Next, collect the table sizes for all original and referenced tables. It is possible that a small number of queries may reference other large tables that will significantly expand the scope of the data to be collected for the benchmark. By eliminating a few sessions, you may be able to dramatically reduce the data size to move to a new platform. Note: Some tables used by the sessions may be volatile or in a work database and no longer have space allocated. Don't eliminate those sessions because they can be adapted with volatile tables.

Step 3: Match the Benchmark Workload to Production Profile

Repeat the profiling process you performed on the production workload with the sessions you've selected so far. You will want to end up with a ratio of tactical, light, medium, and heavy queries that approximate the ratio you had from production. You may have more queries in your sample from Step 2 than you need.

Define a set of 4 to 5 groupings of queries by run seconds or I/O's or number of tables used or a combination of factors from your production workload profiling. Apply those grouping rules to the set of sampled queries to determine how many queries fall into each grouping. To develop the final list of queries, you could execute a number of "insert …. Select …. Where …. Sample n" queries that would produce a sample

	Production	Benchmark
Tactical	100,000	100
Short	20,000	20
Medium	10,000	10
Long	3,000	3

set of sessions with the same ratio as you found in your sampled production day. Ideally you would end up with 50 to 200 sessions of queries.

Now take a final look at the list of tables that will be needed to support the benchmark based on the sessions selected. This will be input to the export step.

Step 4: Adapt Queries for Use in a Benchmark

Data Snapshot Date: 2/09
Query sampled: 3/03
Example Constraints
where INV_DT = current_date-1
where INV_DT = ‘2021-03-02’

The sessions you select may have date constants that made sense when the queries were executed, but the data exported may come from a time period earlier or later than when the queries were run. Other queries may reference “CURRENT_DATE” or “CURRENT_TIMESTAMP” which is not meaningful if the data for the benchmark was exported a month ago. These references will need to be replaced with date literals that are contemporary to the date of the data snapshot taken for the benchmark.

Queries that modify tables need to be identified. There are three situations:

1. If the benchmark query set will be re-run on the production platform, and queries selected make changes to production tables, then those tables will need to be duplicated into alias-named databases. Ensure that the logon IDs used to run the benchmark do not have any access to the production tables to prevent mistakes that could impact production.
2. Queries that modify static tables (production or worktables) will need procedures to reset the modified tables to their original condition after each test. Such sessions can only run once in a test unless provided multiple sets of input data. If DELETE or UPDATE statements are involved, you must restore from a backup copy. If the table maintenance is only inserts, you could either restore from a backup copy or remove the inserted rows based on some indicator or “update date” column in the table.
3. If the queries create temporary tables in a work database, those tables should be converted to volatile tables that are created in the script and dropped at the end. That allows the session to run repeatedly and in parallel across multiple sessions as the concurrency is increased.

Step 5: Snapshot Data for The Benchmark and Export

The list of tables needed comes from the final list of tables from Step 3. Additionally, if any of the sessions will need transaction data to update tables, you should collect sets of that transaction data that were identified in Step 4.

It is critical to check with your Information Security organization before exporting the data. Failure to do so could result in weeks or months of delay when they discover what you are doing. Prepare a plan for dealing with Personally Identifiable Information (PII) and sensitive business data.

- If the columns are merely reported, then they can be replaced with arbitrary strings. (In one case, I developed a list of first and last names and randomly paired them, resulting in some interesting combinations such as "Juan O'Neal". The same approach was used to change the association between house number and street names)

- Columns that are used in constraints need more thoughtful shuffling, respecting the demographics of the column values or the relationships between columns and across tables. For example, random replacement of dates could result in shipments being shipped before they were ordered.
- Columns used for joins need mapping tables so that replacements can be made across multiple tables to maintain referential integrity. The mapping table can be held securely or destroyed after the data is exported.

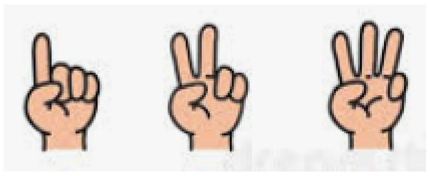
In a 2019 benchmark for an insurance company, they randomly shuffled all dates meaning half of the claims were paid before the incident and sometimes before the policy was in effect. This rendered many queries unusable.

Care must be taken in the export formatting. Too often, text fields include delimiters such as commas or line-end characters making it appear that the input record is ended whereas the next record contains the rest of the columns. Use of quotes or escaping characters both reduces errors and save days or weeks of effort in performing the initial data load. (Remember if using quotes, that the text may have quotes so those must be preceded by an escaping character or doubled, depending on your load utility's input rules.)

Format of exported dates and timestamps should use standard "YYYY-MM-DD" or "YYYY-MM-DD HH:MM:SS" and avoid any suppression of leading zeros. Numbers should be formatted as they are intended to be used, so "2.54" is better than "2.5399999999999998" for floating point numbers.

Step 6 – Select the Mechanism for Running the Benchmark

It is pretty clear that a benchmark won't produce accurate, repeatable results if people in a room are to press the enter key "on the count of three". *(Does that mean pressing when they say three or just after it?)*



Some benchmarks have been run from BI tools which might be ok if the BI tool is the only workload being considered and you are only trying to assess the performance of one new platform (e.g. current on-premises BI server and DBMS server versus moving the DBMS to the cloud). This has some complications if the BI server is being used for other development or production usage or if a new server is to be used, dealing with licensing and installation effort.

If the benchmark is comparing multiple new platforms (a competitive benchmark), then you are measuring more than the differences in your DBMS alternatives. You are also measuring the differences in the network connections and client server overheads which could mask DBMS performance differences.

It is best to choose a query driver that can run queries and ETL jobs in a controlled and repeatable process. We've evolved TdBench through multiple implementations using different technologies over the past 12 years with the objectives of:

- Accurate simulation of customer's actual data warehouse production workload

- Productive setup and execution
- Productive summary and detailed reporting

There are often strong advocates for using J-Meter with a suspicion that TdBench, being provided by a database vendor is biased toward that vendor. If “accurate simulation of a customer’s actual production workload” provides a bias toward Teradata, then … guilty as charged.

J-Meter has a nice GUI and the ability to capture the activity of a user to a server and play back that activity in a test. In a benchmark conducted by one BI tool vendor, they recorded a user running reports at a pace they thought was reasonable, and then played back that activity in an increasing number of sessions of 1, 5, 10, 20, 40 and 80. With J-Meter running the BI tool, we would see bursts of nine to twelve queries fired off at the same time for a single user with gaps of 20-30 seconds between bursts. As concurrency was increased, all sessions would fire queries at approximately the same time causing large fluctuations in actual concurrency.

Advanced users have been able to build JMX files (the definition of the test) outside of the GUI to do some of the same things that TdBench does. Unfortunately, many users lack the expertise to get J-Meter to do a truthful simulation of a production workload.

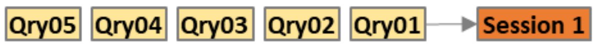
J-Meter	TdBench
JMX defines list of queries to be executed by each thread in a thread pool, but all sessions execute all queries which does not yield a ratio of queries that match production.	Allows simple specification of multiple query queues and a variable number of workers per queue to achieve a production-like query ratio.
Without extra effort, all queries execute in the same order across sessions. With queries executing at the same time in each session, CPU utilization and query production goes down if there is any skew.	TdBench chooses the next query from the queue for each session so concurrent sessions won’t be executing the same query unless there are more sessions than queries.
J-Meter supports a “fixed work” model which as queries across multiple sessions complete, concurrency goes down	TdBench supports both fixed work and fixed period test model. With the fixed period model, queries continue to be initiated for the specified duration of the test.
For prepared queries, J-Meter submits one transaction to prepare the query and another to pass the parameters and repeats for each parameter. This decreases queries per hour in a high latency environment like the cloud.	Each TdBench session issues a prepare for a query once and then re-uses that prepared query for each parameter it executes. This is similar to queries with a USING clause.
J-Meter reports test results it collects from the client which can only answer “What happened in terms of performance?”	TdBench not only collects results in the client but also integrates to DBMS query logging, adding the answer to, “Why it happened?”

TdBench 8.0 is available from Teradata downloads at no cost and supports any DBMS with a JDBC interface. It has been used on benchmarks with Snowflake, Redshift, Greenplum, Oracle, Synapse, Sailfish, Netezza, and Google BigQuery.

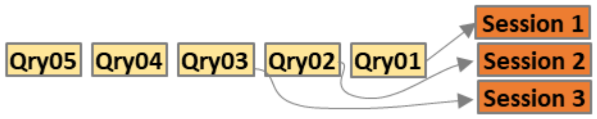
Step 7 –Select Benchmark Test Models

The basics of a query driver is that there is a queue of work to complete and one or more sessions executing the work. There are several queue configurations:

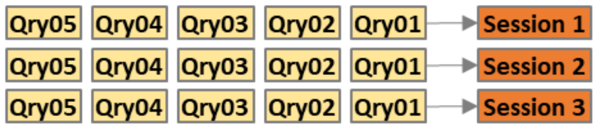
- **Single queue, single worker:** This is also known as a serial test which is designed to understand the performance of each query. It also measures the ability of the DBMS to execute portions of the query in parallel to leverage all computing resources available.



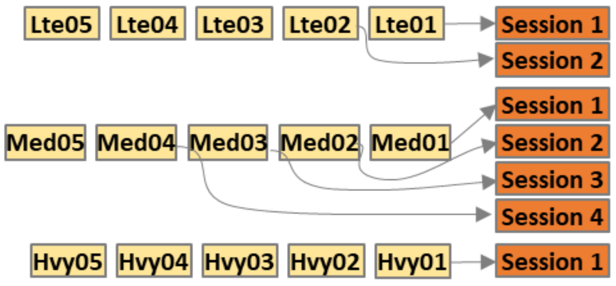
- **Single queue, multi-session:** This is the building block of TdBench where a queue of queries is being processed by multiple worker sessions. As each session completes a query, it retrieves the next query from the queue



- **Multi-queue replicate:** This is the basic way J-Meter handles multi session tests. Each session is going to execute the same list of queries. Worst case, the developer leaves the sequence of all queries in all queues in the same order. Queries that are skewed will tend to skew on the same AMP across all sessions at the same time resulting in under-utilization of the platform. This can be improved with effort to shuffle the queries in each queue. If the test is set to repeat multiple times, each repetition pauses to let each session complete its work, further contributing to under-utilization of the platform.

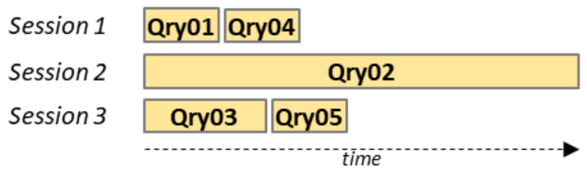


- Multi-queue, multi-session: This allows classifying queries to match the production profile and varying the number of workers per queue to achieve a workload query mix that simulates the production requirements. This model also helps to maintain a constant ratio of heavy, medium and light queries. When there is only one queue with very short and very long and multiple sessions, eventually all of the sessions will be running the long queries. Each queue and its sessions process independently from the others. As sessions complete their queries, they retrieve the next available query from its queue.



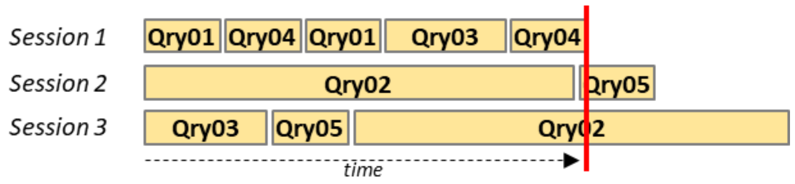
There are several test models that can be used, and different ones are appropriate for different purposes.

- **Fixed Work:** A good use of the fixed work model is the serial test where you have one queue and one session which allows measurement of the best a DBMS can do to fully utilize its resources through parallel process. In some cases, this is used for multi-stream tests and the metric used is the

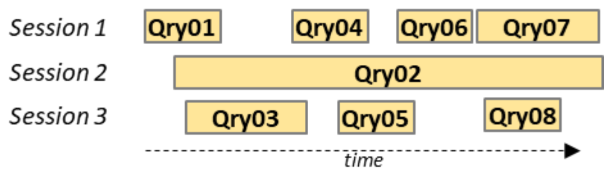


time to complete all queries. The problem is that concurrency is not consistent throughout the test. Basically, you are measuring the longest running query with some competing workload at the beginning of its execution.

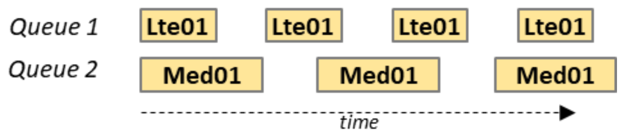
- **Fixed Period Submission:** This model submits queries for a time period and then lets the queries complete. There is consistent concurrency throughout the test.



  This is most useful for comparing the query production capacity of two or more DBMSs. The duration of the test should be set to be long enough for there to be several executions of the longest queries. It is possible to **roughly** calculate partial execution of queries based on the percentage of the query CPU execution that occurred before the end of the fixed period divided by its total CPU in a serial test. For productivity, extremely long queries should not be involved in the workload testing but rather compared purely on the serial testing. It is a good idea to test how short a test you can run and still be able to extrapolate the queries that would be executed in an hour.
- **Fixed Period Completion:** This is similar to the fixed period submission, but at the end of the specified test period, all queries that are in flight are canceled. This allows (e.g.) a test declared to be a half hour to take pretty close to 30 minutes. Otherwise, long running queries could cause the "half hour" test to run 60 to 90 minutes.
- **Query Replay:** This is most useful for validating that a platform can support an existing query arrival rate. If there is adequate capacity, there will be gaps of lower concurrency during the test. If the



  platform can not keep up, it will need more and more sessions to initiate the queries "on time" and if they aren't available, a deficit will be calculated.
- **Paced Interval Query Arrival:** This is most useful for a Proof Of Concept for a new system where there is an assumption on the rate of query arrival. If the platform



  has adequate capacity, queries will be serviced at the intended rate.
- **Paced Percentage Query Arrival:** This is a new feature being added to TdBench 8.01 which allows specification of the percentage ratio of queries between queues. This prevents one queue (e.g. tactical) from running hundreds of thousands of queries on one platform versus an order of magnitude less on another making overall system "Query Per Hour" metrics meaningless.

Step 8 – Design the Benchmark Tests

For comparison between platforms, a standard query test series is:

- Serial Test (Single queue, one session)
- Workload with 5 streams

- Workload with 10 streams
- Workload with 20 streams
- Workload with 40 streams
- Workload with 80 streams.

Begin with a trial serial test and look at the percentage of CPU a single stream uses. If it averages 25%, theoretically, you will be able to run 4 streams before the additional queries impact the query performance. In practice, there will be some impact at 3 concurrent streams, however doubling from 5 to 10 streams will still increase the queries per hour.

On small platforms with large, heavy queries, it may be more appropriate to use a sequence of 1, 2, 4, 8, 16 concurrent streams.

By doubling the number of concurrent streams for each test, it is easy to calculate the expected doubling of response time once you go past the point of CPU or I/O saturation. It is important to remember, once your workload demand has consumed all of an available resource (e.g. CPU), when you double the sessions, that scarce resource will be spread across twice the sessions and with half the resources per session, the queries should take twice as long.

It addition to a test of concurrent queries, it is good to test some update processes as well. First test those update processes serially, then with a moderate level of concurrency such as 10 or 20 query sessions. Each update processes should have a profile of another user being added to the workload.

The update process should be run as a “Fixed Work” test model because repeating the update with the same transactions is not valid. If you captured multiple batches of transactions, then the update process can cycle through the sets of updates, but it should not process the same updates more than once. It is possible in TdBench to schedule the update processes to start at a specified time within a fixed period test and run as a fixed work test even though the query test will continue to cycle for the specified period. If the update process is the focus of the testing and the queries are being used as background workload, they can be set to run continuously and the queue processing updates can terminate all queues when it has finished its fixed period work.

Remember to reset the target tables of the update process after each test.

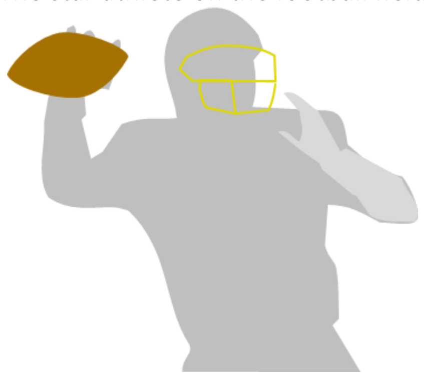
Closing Thoughts

Having built my first enterprise data warehouse in 1975 with multiple subject areas supporting multiple business functions with ad hoc queries, I know that any DBMS you choose can work if you are:

- willing to invest enough effort,
- have the right skills, and
- willing to make enough compromises.

This paper has laid out the steps to construct a benchmark that simulates the requirements of your production system to reduce effort, compromises, and cost… or worse. Just as:

The star athlete on the football field



May fail on the swim team

The DBMS that can run your 20 worst queries or the Industry Standard Benchmarks faster may not be able to keep up with the demands of your production workload which surveys have shown make up 60-90% of the query workload. Running a benchmark that doesn't mimic your real production requirements is as bad as:

- Flipping a coin,
- Drawing names from a hat, or
- Asking for a show of hands

The cost and effort of testing correctly is small compared to the investment and lost opportunity cost if you run meaningless tests and choose incorrectly.