
ClickBench: a Benchmark For Analytical Databases

<https://benchmark.clickhouse.com/>

Discussion: <https://news.ycombinator.com/item?id=32084571>

Overview

This benchmark represents typical workload in the following areas: clickstream and traffic analysis, web analytics, machine-generated data, structured logs, and events data. It covers the typical queries in ad-hoc analytics and real-time dashboards.

The dataset from this benchmark was obtained from the actual traffic recording of one of the world's largest web analytics platforms. It is anonymized while keeping all the essential distributions of the data. The set of queries was improvised to reflect the realistic workloads, while the queries are not directly from production.

Goals

The main goals of this benchmark are:

Reproducibility

You can quickly reproduce every test in as little as 20 minutes (although some systems may take several hours) in a semi-automated way. The test setup is documented and uses inexpensive cloud VMs. The test process is documented in the form of a shell script, covering the installation of every system, loading of the data, running the workload, and collecting the result numbers. The dataset is published and made available for download in multiple formats.

Compatibility

The tables and queries use mostly standard SQL and require minimum or no adaptation for most SQL DBMS. The dataset has been filtered to avoid difficulties with parsing and loading.

Diversity

The benchmark process is easy enough to cover a wide range of systems. It includes: modern and historical self-managed OLAP DBMS; traditional OLTP DBMS are included for comparison baseline;

managed database-as-a-service offerings are included, as well as serverless cloud-native databases; some NoSQL, document, and specialized time-series databases are included as well for a reference, even if they should not be comparable on the same workload.

Realism

The dataset is derived from accurate production data. The realistic data distributions allow for correctly accounting for compression, indices, codecs, custom data structures, etc., which is not possible with most of the random dataset generators. The workload consists of 43 queries and can test the efficiency of full scan and filtered scan, as well as index lookups, and main relational operations. It can test various aspects of hardware as well: some queries require high storage throughput; some queries benefit from a large number of CPU cores, and some benefit from single-core speed; some queries benefit from high main memory bandwidth.

Limitations

The limitations of this benchmark allow keeping it easy to reproduce and to include more systems in the comparison. The benchmark represents only a subset of all possible workloads and scenarios. While it aims to be as fair as possible, focusing on a specific subset of workloads may give an advantage to the systems that specialise in those workloads.

The following limitations should be acknowledged:

1. The dataset is represented by one flat table. This is not representative of classical data warehouses, which use a normalized star or snowflake data model. The systems for classical data warehouses may get an unfair disadvantage on this benchmark.
2. The table consists of exactly 99 997 497 records. This is rather small by modern standards but allows tests to be performed in a reasonable time.
3. While this benchmark allows testing distributed systems, and it includes multi-node and serverless cloud-native setups, most of the results so far have been obtained on a single node setup.
4. The benchmark runs queries one after another and does not test a workload with concurrent requests; neither does it test for system capacity. Every query is run only a few times, and this allows some variability in the results.
5. Many setups and systems are different enough to make direct comparison tricky. It is not possible to test the efficiency of storage used for in-memory databases, or the time of data loading for stateless query engines. The goal of the benchmark is to give the numbers for comparison and let you derive the conclusions on your own.

TLDR: *All Benchmarks Are Bastards Liars.*

Rules and Contribution

How To Add a New Result

To introduce a new system, simply copy-paste one of the directories and edit the files accordingly: - `benchmark.sh`: this is the main script to run the benchmark on a fresh VM; Ubuntu 22.04 or newer should be used by default, or any other system if specified in the comments. The script may not necessarily run in a fully automated manner - it is recommended always to copy-paste the commands one by one and observe the results. For managed databases, if the setup requires clicking in the UI, write a `README.md` instead. - `README.md`: contains comments and observations if needed. For managed databases, it can describe the setup procedure to be used instead of a shell script. - `create.sql`: a CREATE TABLE statement. If it's a NoSQL system, another file like `wtf.json` can be presented. - `queries.sql`: contains 43 queries to run; - `run.sh`: a loop for running the queries; every query is run three times; if it's a database with local on-disk storage, the first query should be run after dropping the page cache; - `results`: put the .json files with the results for every hardware configuration there.

To introduce a new result for an existing system on different hardware configurations, add a new file to `results`.

To introduce a new result for an existing system with a different usage scenario, either copy the whole directory and name it differently (e.g. `timescaledb`, `timescaledb-compression`) or add a new file to the `results` directory.

Installation And Fine-Tuning

The systems can be installed or used in any reasonable way: from a binary distribution, from a Docker container, from the package manager, or compiled - whatever is more natural and simple or gives better results.

It's better to use the default settings and avoid fine-tuning. Configuration changes can be applied if it is considered strictly necessary and documented.

Fine-tuning and optimization for the benchmark are not recommended but allowed. In this case, add the results on vanilla configuration and fine-tuned configuration separately.

Data Loading

The dataset is available in [CSV](#), [TSV](#), [JSONlines](#) and [Parquet](#) formats by the following links:

- https://datasets.clickhouse.com/hits_compatible/hits.csv.gz
- https://datasets.clickhouse.com/hits_compatible/hits.tsv.gz
- https://datasets.clickhouse.com/hits_compatible/hits.json.gz
- https://datasets.clickhouse.com/hits_compatible/hits.parquet

You can select the dataset format at your convenience.

Additional sources for stateless table engines are provided: - https://datasets.clickhouse.com/hits_compatible/athena (the same parquet file in its own subdirectory) - https://datasets.clickhouse.com/hits_compatible/athena_partitioned (100 files)

To correctly compare the insertion time, the dataset should be downloaded and decompressed before loading (if it's using external compression; the parquet file includes internal compression and can be loaded as is). The dataset should be loaded as a single file in the most straightforward way. Splitting the dataset for parallel loading is not recommended, as it will make comparisons more difficult. Splitting the dataset is possible if the system cannot eat it as a whole due to its limitations.

You should not wait for cool down after data loading or running OPTIMIZE / VACUUM before the main benchmark queries unless it is strictly required for the system.

The used storage size can be measured without accounting for temporary data if there is temporary data that will be removed in the background. The built-in introspection capabilities can be used to measure the storage size, or it can be measured by checking the used space in the filesystem.

Indexing

The benchmark table has one index - the primary key. The primary key is not necessary to be unique. The index of the primary key can be made clustered (ordered, partitioned, sharded).

Manual creation of other indices is not recommended, although if the system creates indexes automatically, it is considered ok.

Preaggregation

The creation of pre-aggregated tables or indices, projections, or materialized views is not recommended for the purpose of this benchmark. Although you can add fine-tuned setup and results for reference, they will be out of competition.

If a system is of a “multidimensional OLAP” kind, and so is always or implicitly doing aggregations, it can be added for comparison.

Caching

If the system contains a cache for query results, it should be disabled.

It is okay if the system performs caching for source data (buffer pools and similar). If the cache or buffer pools can be flushed, they should be flushed before the first run of every query.

If the system contains a cache for intermediate data, that cache should be disabled if it is located near the end of the query execution pipeline, thus similar to a query result cache.

Incomplete Results

Many systems cannot run the full benchmark suite successfully due to OOMs, crashes, or unsupported queries. The partial results should be included nevertheless. Put `null` for the missing numbers.

If The Results Cannot Be Published

Some vendors don’t allow publishing benchmark results due to the infamous DeWitt Clause. Most of them still allow the use of the system for benchmarks. In this case, please submit the full information about installation and reproduction, but without the `results` directory. A `.gitignore` file can be added to prevent accidental publishing.

We allow both open-source and proprietary systems in our benchmark, as well as managed services, even if registration, credit card, or salesperson call is required - you still can submit the testing description if you don’t violate the TOS.

Please let us know if some results were published by mistake by opening an issue on GitHub.

If a Mistake Or Misrepresentation Is Found

It is easy to accidentally misrepresent some systems. While acting in good faith, the authors admit their lack of deep knowledge of most systems. Please send a pull request to correct the mistakes.

Results Usage And Scoreboards

The results can be used for comparison of various systems, but always take them with a grain of salt due to the vast amount of caveats and hidden details. Always reference the original benchmark and this text.

We allow but do not recommend creating scoreboards from this benchmark or saying that one system is better (faster, cheaper, etc.) than another.

There is a web page to navigate across benchmark results and present a summary report. It allows filtering out some systems, setups, or queries. For example, if you found some subset of the 43 queries are irrelevant, you can simply exclude them from the calculation and share the report without these queries.

You can select the summary metric from one of the following: “Cold Run”, “Hot Run”, “Load Time”, and “Data Size”. If you select the “Load Time” or “Data Size”, the entries will be simply ordered from best to worst, and additionally, the ratio to the best non-zero result will be shown (the number of times one system is worse than the best system in this metric). Load time can be zero for stateless query engines like [clickhouse-local](#) or [Amazon Athena](#).

If you select “Cold Run” or “Hot Run”, the aggregation across the queries is performed in the following way:

1. The first run for every query is selected for Cold Run. For Hot Run, the minimum from 2nd and 3rd run time is selected, if both runs are successful, or null if some were unsuccessful.

By default, the “Hot Run” metric is selected, because it’s not always possible to obtain a cold runtime for managed services, while for on-premise a quite slow EBS volume is used by default which makes the comparison slightly less interesting.

2. For every query, find a system that demonstrated the best (fastest) query time and take it as a baseline.

This gives us a point of comparison. Alternatively, we can take a benchmark entry like “ClickHouse on c6a.metal” as a baseline and divide all query times by the baseline time. This would be quite arbitrary and asymmetric. Instead, we take the best result for every query separately.

3. For every query, if the result is present, calculate the ratio to the baseline, but add constant 10ms to the nominator and denominator, so the formula will be: $(10\text{ms} + \text{query_time}) / (10\text{ms} + \text{baseline_query_time})$. This formula gives a value ≥ 1 , which is equal to 1 for the best benchmark entry on this query.

We are interested in relative query run times, not absolute. The benchmark has a broad set of queries, and there can be queries that typically run in 100ms (e.g., for interactive dashboards) and some

queries that typically run in a minute (e.g., complex ad-hoc queries). And we want to treat these queries as equally important in the benchmark, that's why we need relative values.

The constant shift is needed to make the formula well-defined when query time approaches zero. For example, some systems can get query results in 0 ms using table metadata lookup, and another in 10 ms by range scan. But this should not be treated as the infinite advantage of one system over the other. With the constant shift, we will treat it as only two times an advantage.

4. For every query, if the result is not present, substitute it with a “penalty” calculated as follows: take the maximum query runtime for this benchmark entry across other queries that have a result, but if it is less than 300 seconds, put it 300 seconds. Then multiply the value by 2. Then calculate the ratio as explained above.

For example, one system crashed while trying to run a query which can highlight the maturity, or lack of maturity, of a system. Or does not run a query due to limitations. If this system shows run times like 1..1000 sec. on other queries, we will substitute 2000 sec. instead of this missing result.

5. Take the geometric mean of the ratios across the queries. It will be the summary rating.

Why geometric mean? The ratios can only be naturally averaged in this way. Imagine there are two queries and two systems. The first system ran the first query in 1s and the second query in 20s. The second system ran the first query in 2s and the second query in 10s. So, the first system is two times faster on the first query and two times slower on the second query and vice-versa. The final score should be identical for these systems.

History and Motivation

The benchmark was created in October 2013 to evaluate various DBMS to use for a web analytics system. It has been made by taking 1/50th of one week of production pageviews (a.k.a. “hits”) data and taking the first one billion, one hundred million, and ten million records from it. It has been run on a 3-node cluster of Xeon E2650v2 with 128 GiB RAM, 8x6TB HDD in md-RAID-6, and 10 Gbit network in a private datacenter in Finland.

The following systems were tested in 2013: ClickHouse, MonetDB, InfiniDB, Infobright, LucidDB, Vertica, Hive and MySQL. To ensure fairness, the benchmark has been conducted by a person without ClickHouse experience. ClickHouse has been selected for production usage by the results of this benchmark.

The benchmark continued to be occasionally used privately until 2016 when the results were published with the ClickHouse release in open-source. While the results were made public, the datasets were not, as they contain customer data.

We needed to publish the dataset to facilitate open-source development and testing, but it was not possible to do it as is. In 2019, the [clickhouse-obfuscator](#) tool was introduced to anonymize the data, and the dataset was published. Read more about the challenge of data obfuscation [here](#).

More systems were included in the benchmark over time: Greenplum, MemSQL (now SingleStore), OmniSci (now HeavyAI), DuckDB, PostgreSQL, and TimescaleDB.

In 2021 the original cluster for benchmark stopped being used, and we were unable to add new results without rerunning the old results on different hardware. Rerunning the old results appeared to be difficult: due to the natural churn of the software, the old step-by-step instructions become stale.

The original benchmark dataset included many details that were natural for ClickHouse and web analytics data but hard for other systems: unsigned integers (not supported by standard SQL), strings with zero bytes, fixed-length string data types, etc. Only ClickHouse was able to load the dataset as is, while most other databases required non-trivial adjustments to the data and queries.

The idea of the new benchmark is: - normalize the dataset to a “common denominator”, so it can be loaded to most of the systems without a hassle. - normalize the queries to use only standard SQL - they will not use any advantages of ClickHouse but will be runnable on every system. - ideally make it automated. At least make it simple - runnable by a short shell script that can be run by copy-pasting a few commands in the terminal, in the worst case. - run everything on widely available cloud VMs and allow recording the results from various types of instances.

The benchmark is created and used by the ClickHouse team. It can be surprising, but we did not perform any specific optimizations in ClickHouse for the queries in the benchmark, which allowed us to keep some reasonable sense of fairness with respect to other systems.

Now the new benchmark is easy to use and the results for any system can be reproduced in around 20 minutes.

We also introduced the Hardware Benchmark for testing servers and VMs.

Systems Included

- ☒ ClickHouse
- ☒ ClickHouse on local Parquet files
- ☒ ClickHouse operating like “Athena” on remote Parquet files
- ☒ ClickHouse on a VFS over HTTPs on CDN
- ☒ MySQL InnoDB
- ☒ MySQL MyISAM
- ☒ MariaDB
- ☒ MariaDB ColumnStore

-
- ☒ MemSQL/SingleStore
 - ☒ PostgreSQL
 - ☒ Greenplum
 - ☒ TimescaleDB
 - ☒ Citus
 - ☒ Vertica (without publishing)
 - ☒ QuestDB
 - ☒ DuckDB
 - ☒ DuckDB over local Parquet files
 - ☐ DuckDB operating like “Athena” on remote Parquet files
 - ☒ MonetDB
 - ☒ mapD/Omnisci/HeavyAI
 - ☒ Databend
 - ☒ DataFusion
 - ☒ ByteHouse
 - ☒ Doris/PALO
 - ☒ SelectDB
 - ☒ Druid
 - ☒ Pinot
 - ☒ CrateDB
 - ☐ Spark SQL
 - ☒ Starrocks
 - ☐ ShitholeDB
 - ☐ Hive
 - ☒ Hydra
 - ☐ Impala
 - ☐ Hyper
 - ☒ Umbra
 - ☒ SQLite
 - ☒ Redshift
 - ☒ Redshift Serverless
 - ☐ Redshift Spectrum
 - ☐ Presto
 - ☐ Trino
 - ☒ Amazon Athena
 - ☒ Bigquery (without publishing)
 - ☒ Snowflake
 - ☐ Rockset
-

-
- ☐ CockroachDB
 - ☐ CockroachDB Serverless
 - ☐ Databricks
 - ☐ Planetscale (without publishing)
 - ☐ TiDB (TiFlash)
 - ☒ Amazon RDS Aurora for MySQL
 - ☒ Amazon RDS Aurora for Postgres
 - ☐ InfluxDB
 - ☐ TDEngine
 - ☒ MongoDB
 - ☐ Cassandra
 - ☐ ScyllaDB
 - ☒ Elasticsearch
 - ☐ Apache Ignite
 - ☒ Motherduck
 - ☒ Infobright
 - ☐ Actian Vector
 - ☐ Manticore Search
 - ☒ Vertica (without publishing)
 - ☐ Azure Synapse
 - ☐ Starburst Galaxy
 - ☐ MS SQL Server with Column Store Index (without publishing)
 - ☐ Dremio (without publishing)
 - ☐ Exasol
 - ☐ LocustDB
 - ☐ EventQL
 - ☐ Apache Drill
 - ☐ Apache Kudu
 - ☐ Apache Kylin
 - ☐ S3 select command in AWS
 - ☒ Kinetica
 - ☐ YDB
 - ☐ OceanBase
 - ☐ Boilingdata
 - ☒ Byteconity
 - ☐ DolphinDB
 - ☒ Osla
 - ☐ Quickwit

-
- ☒ AlloyDB
 - ☒ ParadeDB
 - ☒ GlareDB
 - ☐ Seafoal
 - ☐ Sneller
 - ☒ Tablespace
 - ☒ Tembo

By default, all tests are run on c6a.4xlarge VM in AWS with 500 GB gp2.

Please help us add more systems and run the benchmarks on more types of VMs.

Similar Projects

Many alternative benchmarks are applicable to OLAP DBMS with their own advantages and disadvantages.

Brown University Mgbench

<https://github.com/crotyan/mgbench>

A new analytical benchmark for machine-generated log data. By Andrew Crottyan from Brown University.

Advantages: - somewhat realistic dataset; - a diverse set of queries; - good coverage of systems; - easy to reproduce;

Disadvantages: - very small dataset size; - favors in-memory databases; - mostly abandoned.

UC Berkeley AMPLab Big Data Benchmark

<https://amplab.cs.berkeley.edu/benchmark/>

Poor coverage of queries that are too simple. The benchmark is abandoned.

Mark Litwinski's NYC Taxi

<https://tech.marksblogg.com/benchmarks.html>

Advantages: - real-world dataset; - good coverage of systems; many unusual entries; - contains a story for every benchmark entry;

Disadvantages: - unreasonably small set of queries: 4 mostly trivial queries don't represent any realistic workload and are subjects for over-optimization; - compares different systems on different hardware; - many results are outdated; - no automated or easy way to reproduce the results; - while many results are performed independently of corporations or academia, some benchmark entries may have been sponsored; - the dataset is not readily available for downloads: originally 1.1 billion records are used, while it's more than 4 billion records in 2022.

Database-like ops Benchmark from h2o.ai

<https://h2oai.github.io/db-benchmark/>

A benchmark for data-frame libraries and embedded databases. Good coverage of data-frame libraries and a few full-featured DBMS as well.

A benchmark for querying large JSON datasets

https://colab.research.google.com/github/dcmoura/spyql/blob/master/notebooks/json_benchmark.ipynb

A good benchmark for command-line tools for processing semistructured data. Can be used to test DBMS as well.

Star Schema Benchmark

Pat O'Neil, Betty O'Neil, Xuedong Chen <https://www.cs.umb.edu/~poneil/StarSchemaB.PDF>

It is a simplified version of TPC-H.

Advantages: - well-specified; - popular in academia;

Disadvantages: - represents a classic data warehouse schema; - database generator produces random distributions that are not realistic and the benchmark does not allow for the capture of differences in various optimizations that matter on real-world data; - many research systems in academia targeting for this benchmark which makes many aspects of it exhausted;

TPC-H

A benchmark suite from Transaction Processing Council - one of the oldest organizations specializing in DBMS benchmarks.

Advantages: - well-specified;

Disadvantages: - requires official certification; - represents a classic data warehouse schema; - database generator produces random distributions that are not realistic and the benchmark does not allow for the capture of differences in various optimizations that matter on real-world data; - many systems are targeting this benchmark which makes many aspects of it exhausted;

TPC-DS

More advanced than TPC-H, focused on complex ad-hoc queries. This also requires official certification.

Advantages: - an extensive collection of complex queries.

Disadvantages: - requires official certification; - official results have only sparse coverage of systems; - biased towards complex queries over many tables.

Ontime

Introduced by Vadim Tkachenko from Percona in 2009.

Based on the US Bureau of Transportation Statistics open data.

Advantages: - real-world dataset;

Disadvantages: - not widely used; - the set of queries is not standardized; - the table contains too much redundancy;

TSBS

Time Series Benchmark Suite. <https://github.com/timescale/tsbs> Originally from InfluxDB, and supported by TimescaleDB.

Advantages: - a benchmark for time-series scenarios;

Disadvantages: - not applicable for scenarios with data analytics.

Fair Database Benchmarks

<https://github.com/db-benchmarks/db-benchmarks>

A benchmark suite inspired by ClickHouse benchmarks. Used mostly to compare search engines: Elasticsearch and Manticore.

SciTS

<https://arxiv.org/abs/2204.09795> or <https://dl.acm.org/doi/10.1145/3538712.3538723> A new benchmark for time-series workloads.

Tests both insertion and query speeds, as well as resource consumption.

Includes TimescaleDB, InfluxDB, PostgreSQL and ClickHouse.

This benchmark is fully independent and open-source.

STAC

<https://www.stacresearch.com/>

Disadvantages: - requires a paid membership.

More...

Please let me know if you know more well-defined, realistic, and reproducible benchmarks for analytical workloads.

In addition, I collect every benchmark that includes ClickHouse here.

Additional Outcomes

This benchmark can be used to collect the snippets for installation and data loading across a wide variety of DBMS. The usability and quality of the documentation can be compared. It has been used to improve the quality of the participants as demonstrated in [duckdb#3969](#), [timescaledb#4473](#), [mariadb-corporation#16](#), [MonetDB#7309](#), [questdb#2272](#), [crate#12654](#), [LocustDB#152](#), [databend#9738](#), [databend#9612](#), [databend#10226](#), [databend#10195](#), [databend#9978](#), [databend#9965](#), [databend#9809](#), [databend#9716](#), [databend#9600](#), [databend#9565](#) etc. ### References and Citation

Alexey Milovidov, 2022.