ClickHouse Features That I Like

— My favorite features

— Late 2022 and early 2023

Parameterized Views

```
CREATE VIEW wiki
AS SELECT toStartOfMonth(time),
          sum(hits) AS h,
          bar(h, 0, max(h) OVER (), 100)
    FROM wikistat
    WHERE path = {page:String}
    GROUP BY 1
    ORDER BY 1;

SELECT * FROM wiki(page = 'ClickHouse');
```

Developer: Smita Kulkarni.

Regexp Tree Dictionaries

```
CREATE DICTIONARY user_agent
(
    regexp String,
    name String,
    version UInt16
)
PRIMARY KEY(regexp)
SOURCE(YAMLRegExpTree(PATH '/.../regexp_tree.yaml'))
LAYOUT(regexp_tree)
```

Developers: Vage Ogannisian, Han Fei.

Regexp Tree Dictionaries

```
- regexp: 'Linux/(\d+[\.\d]*).+tlinux'
  name: 'TencentOS'
  version: '\1'

- regexp: '\d+/tclwebkit(?:\d+[\.\d]*)'
  name: 'Andriod'
  versions:
    - regexp: '33/tclwebkit'
      version: 13
    - regexp: '3[12]/tclwebkit'
      version: 12
    - regexp: '30/tclwebkit'
      version: 11
    - regexp: '29/tclwebkit'
      version: 10
```

Regexp Tree Dictionaries

```
SELECT dictGet('user_agent', ('name', 'version'), UserAgent);
```

— Traverses the tree, and determines the values of the attributes.

— The tree can be arbitrarily deep,
and each nested level can override the values.

```
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/109.0.0.0 Safari/537.36
```

— All regular expressions are checked in a single pass for performance!

— Can be loaded from YAML file or from a table in any source.

Query Result Cache

```
$ cat /etc/clickhouse-server/config.d/query_cache.yaml
query_result_cache:
    size: 1073741824
    max_entries: 1024
    max_entry_size: 104857600
    max_entry_records: 30000000
```

```
SET enable_experimental_query_result_cache = 1;
```

Developers: Mikhail Stetsyuk, Robert Schulze.

Query Result Cache

Allows to control on per-query basis:

— min query runs to cache the result;
— min query runtime to cache the result;
— max result size to put into cache;
— max staleness to use the cached entry;
— passive usage of the existing entries in cache;
— caching of queries with non-deterministic functions;
— sharing the cache between users;

Developers: Mikhail Stetsyuk, Robert Schulze.

Next steps: compressed cache; on disk cache; cache of intermediate data.

Inverted Full Text Indices

```
ALTER TABLE hackernews_indexed
  ADD INDEX inv_idx(text) TYPE inverted;

ALTER TABLE hackernews_indexed MATERIALIZE INDEX inv_idx;
```

Supports tokens and n-grams.

Optimizes **hasToken**, **multiSearchAny**, equality comparison...

Developer: Larry Luo, Harry Lee, Robert Schulze.

Inverted Full Text Indices

```
:) SELECT count() FROM hackernews WHERE hasToken(text, 'ClickHouse')

┌─count()─┐
│     948 │
└─────────┘

1 row in set. Elapsed: 0.579 sec. Processed 33.95 million rows,
11.61 GB (58.63 million rows/s., 20.05 GB/s.)

:) SELECT count() FROM hackernews_indexed WHERE hasToken(text, 'ClickHouse')

┌─count()─┐
│     948 │
└─────────┘

1 row in set. Elapsed: 0.206 sec. Processed 3.71 million rows,
1.32 GB (18.00 million rows/s., 6.43 GB/s.)
```

Developer: Larry Luo, Harry Lee.

Backup & Restore

— full and incremental backups;

— tables, partitions, databases, all databases;

— full or partial restore;

— a bunch of files or a single archive;

— atomic snapshot for MergeTree,
best effort snapshot for multiple tables;

Developer — Vitaliy Baranov.

Backup & Restore

```
BACKUP TABLE t TO File('backup_20220629');

BACKUP TABLE t TO File('backup_20220629.zip');

BACKUP TABLE t TO File('backup_20220630')
    SETTINGS base_backup = File('backup_20220629');

BACKUP TABLE system.users, system.grants TO ...
BACKUP TABLE system.functions TO ...
```

Backup & Restore

```
BACKUP|RESTORE
  TABLE [db.]table_name [AS [db.]table_name_in_backup]
    [PARTITION[S] partition_expr [,...]] |
  DICTIONARY [db.]dictionary_name [AS [db.]name_in_backup] |
  DATABASE database_name [AS database_name_in_backup]
    [EXCEPT TABLES ...] |
  TEMPORARY TABLE table_name [AS table_name_in_backup] |
  ALL TEMPORARY TABLES [EXCEPT ...] |
  ALL DATABASES [EXCEPT ...] } [,...]
  [ON CLUSTER 'cluster_name']
  TO|FROM File('path/') | Disk('disk_name', 'path/') | S3(...)
  [SETTINGS base_backup = File(...) | Disk(...)]
```

Reliability

ClickHouse should always work.

Asynchronous Initialization Of Tables

If ZooKeeper is unavailable at server startup,
the **ReplicatedMergeTree** tables will start in read-only mode
and initialize asynchronously in background
as soon as ZooKeeper will be available.

— the same applicable for ClickHouse Keeper as well;

— especially useful for embedded ClickHouse Keeper;

Developers: Antonio Andelic. Since 22.9.

Retries On INSERT

Session expired. Table is in readonly mode ☹

Never again:

```
SET insert_keeper_max_retries = 10;
```

INSERT will survive restarts of ClickHouse Keeper or ZooKeeper
and reconnections.

Developer: Igor Nikonov. Since 22.11.

Faster Reconnection to Keeper

Table is in readonly mode 😡

Was: around one minute for reconnection, for no reason.

Now: milliseconds 🥲

Developer: Raul Marin. Since 22.10.

Flexible Memory Limits

Was: strict per-query limit, **max_memory_usage** = 10 GB by default.

Now: query can use the memory if it's available;
in case of memory shortage, the most "overcommitted" query is terminated with exception.

Developer: Dmitry Novik.

SQL Compatibility

ClickHouse should support everything you expect.

Analyzer

```
WITH example AS (
    SELECT '2021-01-01' AS date,
        1 AS node, 1 AS user)
SELECT extra_data FROM (
    SELECT join1.*
    FROM example
    LEFT JOIN (
        SELECT '2021-01-01' AS date,
            1 AS extra_data) AS join1
    ON example.date = join1.date
    LEFT JOIN (
        SELECT '2021-01-01' AS date) AS join2
    ON example.date = join2.date)
```

Was: Missing columns: 'extra_data' while processing query...

Now: **SET allow_experimental_analyzer = 1;**
- works perfectly.

Analyzer

```
SELECT * FROM (SELECT SUM(COALESCE(SEQ_VONR_MO_CALL_CONN_FAIL_TIMES_C, 0)) AS VONR_MO_CALL_CONN_FAIL_TIMES,
MT.`102520001` AS `102520001`, MT.`181361814368` AS `181361814368`, MT.`102520102` AS `102520102`, MT.`102520101`
AS `102520101`, MT.`102520104` AS `102520104`, MT.`183111861371` AS `183111861371`, MT.`102530101` AS
`102530101`, MT.`102540101` AS `102540101`, MT.`102520103` AS `102520103`, MT.`102510101` AS `102510101` FROM (
SELECT COALESCE(SUM(VONR_MO_CALL_CONN_FAIL_TIMES), 0) AS SEQ_VONR_MO_CALL_CONN_FAIL_TIMES_C, COM_FAIL_CAUSE AS
`102520001`, NULL AS `102520102`, COM_FAIL_CAUSE AS `102510101`, NULL AS `102520101`,
D183111570684_H101.`183111861371` AS `183111861371`, NULL AS `102520104`, NULL AS `102520103`,
H_COMPREHENSIVE_FAILURE_CAUSE.`102540101` AS `102540101`, H_COMPREHENSIVE_FAILURE_CAUSE.`102530101` AS
`102530101`, concat('14', '-', '255', '-', '255', '-', SIP_RELEASE_CODE) AS `181361814368` FROM
TEST_DATABASE.SDR_TEST LEFT JOIN ( SELECT concat(PROTOCOL_ID, '-', FIRFAILMSG_ID, '-', PD_ID, '-', CAUSE_ID) AS
`102510101`, UNIFIED_CAUSE_ID AS `183111861371`, concat(FAILCAUSE, '(', PD, ')') AS NAME_102510101 FROM
TEST_DATABASE.DIM_TEST GROUP BY concat(PROTOCOL_ID, '-', FIRFAILMSG_ID, '-', PD_ID, '-', CAUSE_ID),
UNIFIED_CAUSE_ID, concat(FAILCAUSE, '(', PD, ')') ) AS D183111570684_H101 ON COM_FAIL_CAUSE =
D183111570684_H101.`102510101` LEFT JOIN ( SELECT concat(PROTOCOL_ID, '-', FIRFAILMSG_ID, '-', PD_ID, '-',
CAUSE_ID) AS `102520001`, SCENE_ID AS `102540101`, CLASS_ID AS `102530101`, SCENE_NAME AS NAME_102540101 FROM
TEST_DATABASE.DIM_TEST GROUP BY concat(PROTOCOL_ID, '-', FIRFAILMSG_ID, '-', PD_ID, '-', CAUSE_ID), SCENE_ID,
CLASS_ID, SCENE_NAME ) AS H_COMPREHENSIVE_FAILURE_CAUSE ON COM_FAIL_CAUSE =
H_COMPREHENSIVE_FAILURE_CAUSE.`102520001` LEFT JOIN ( SELECT concat(PROTOCOL_ID, '-', FIRFAILMSG_ID, '-', PD_ID,
'-', CAUSE_ID) AS `181361814368`, CAUSE AS NAME_181361814368 FROM TEST_DATABASE.DIM_TEST GROUP BY
concat(PROTOCOL_ID, '-', FIRFAILMSG_ID, '-', PD_ID, '-', CAUSE_ID), CAUSE ) AS DIM_FAILCAUSE_ALL_181361814368 ON
concat('14', '-', '255', '-', '255', '-', SIP_RELEASE_CODE) = DIM_FAILCAUSE_ALL_181361814368.`181361814368` WHERE
(H_COMPREHENSIVE_FAILURE_CAUSE.NAME_102540101 IS NOT NULL) AND (D183111570684_H101.NAME_102510101 IS NOT NULL)
AND (DIM_FAILCAUSE_ALL_181361814368.NAME_181361814368 IS NOT NULL) GROUP BY `102520001`, `102520102`,
`102510101`, `102520101`, D183111570684_H101.`183111861371`, `102520104`, `102520103`,
H_COMPREHENSIVE_FAILURE_CAUSE.`102540101`, H_COMPREHENSIVE_FAILURE_CAUSE.`102530101`, `181361814368` ) AS MT
GROUP BY MT.`102520001`, MT.`181361814368`, MT.`102520102`, MT.`102520101`, MT.`102520104`, MT.`183111861371`,
MT.`102530101`, MT.`102540101`, MT.`102520103`, MT.`102510101` ) AS ST WHERE ST.VONR_MO_CALL_CONN_FAIL_TIMES > 0
ORDER BY VONR_MO_CALL_CONN_FAIL_TIMES DESC LIMIT 0, 5000
```

Window Functions Inside Expressions

```
SELECT
    y::String || '.'
        || (y < toYear(today()) - 2000 - 1 ? '*' : m::String) AS Version,
    (n <= 3 OR (is_lts AND lts_n <= 2)) ? '✔' : '❌' AS Supported
FROM (
    SELECT y, m,
        count() OVER (ORDER BY y DESC, m DESC) AS n,
        m IN (3, 8) AS is_lts,
        countIf(is_lts) OVER (ORDER BY y DESC, m DESC) AS lts_n
    FROM (
        WITH extractGroups(version, 'v(\d+)\.(\d+)') AS v,
             v[1]::INT AS y, v[2]::INT AS m
        SELECT y, m
        FROM file('version_date.tsv', TSV, 'version String, date String')
        ORDER BY y DESC, m DESC
        LIMIT 1 BY y, m)
)
LIMIT 1 BY Version
FORMAT Markdown
```

DELETE Query

```
SET mutations_sync = 1;
ALTER TABLE hits
DELETE WHERE Title LIKE '%Mongo%';
```

— 205 sec (for a table with 100 million records).

```
DELETE FROM hits
WHERE Title LIKE '%Mongo%';
```

— ??? sec.

Developers: Alexander Gololobov, Jianmei Zhang.

Non-Constant LIKE and match

```
SELECT DISTINCT repo_name, title
FROM github_events
WHERE title ILIKE (
  repo_name LIKE '%ClickHouse%' ? '%fast%' : '%slow%')
AND repo_name IN ('ClickHouse/ClickHouse', 'elastic/elasticsearch')
```

Now I can put LIKE inside LIKE and looks like you're going to like it.

Developer: Robert Schulze. Since 22.6.

Performance

ClickHouse never slows down!

Performance Optimizations

Speed-up of SELECT with **FINAL** modifier.

It "simply" improves performance up to 4 times.

Especially for complex transforms like Collapsing and Replacing.

Developer: Nikita Taranov.

More Performance

Optimize **COUNT(DISTINCT ...)** for low number of GROUP BY keys.

Optimize **GROUP BY** with CPU prefetcher.

Optimize **GROUP BY** with better block sizes.

Developer: Nikita Taranov.

New JOIN algorithms

— "**direct**" algorithm:
to join with key-value tables by direct lookups a.k.a. nested loops.
Good if the left table is small, but the right table is like a large dictionary.
Good to use in MATERIALIZED VIEW queries.

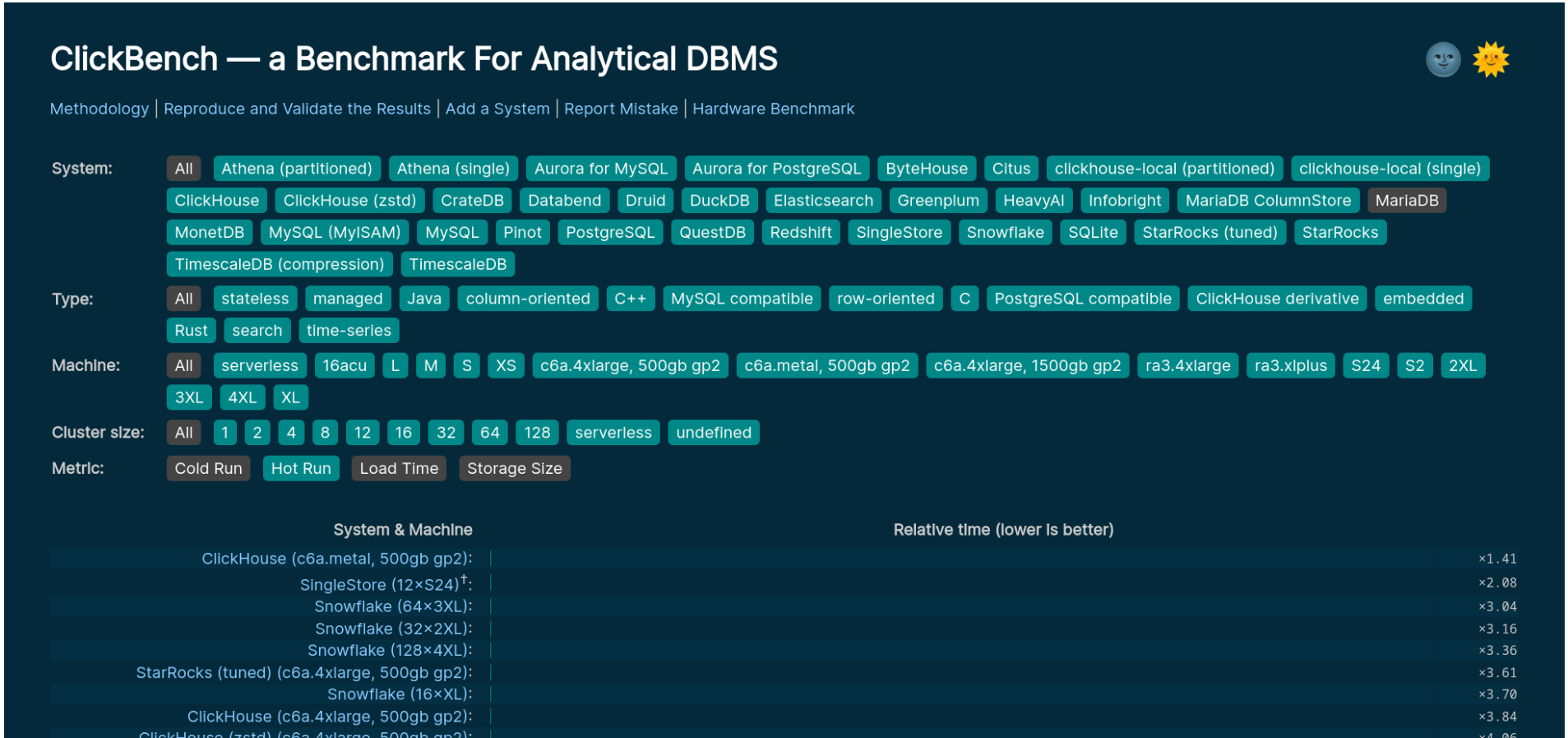
— "**parallel_hash**" algorithm:
speed-up if the right hand table is large.

— "**full_sorting_merge**" algorithm:
when right hand side is large
and does not fit in memory and does not allow lookups.

— "**grace_hash**" algorithm:
since in 22.12.

Developer: Vladimir Cherkasov, lgbo-ustc.

Updated Benchmark



Integrations

ClickHouse can work **as a server** (clickhouse-server)
or **as a tool** without installation (clickhouse-local).

ClickHouse can **store the data**
or process externally stored data **on the fly**.

External data:
— remote databases: MySQL, PostgreSQL, MongoDB, ODBC, JDBC...
— object storages: S3, HDFS, Azure, COSN, OSS...
— from URL and local files;

All possible data formats:
— text: CSV, TSV, JSON, Values, MySQLDump, Regexp...
— binary: Parquet, Arrow, ORC, Avro, Protobuf, MsgPack...
— schemaful and schemaless;

Data Lakes

Now ClickHouse supports **Apache Hudi**, **Delta Lake**, and **Iceberg** for SELECT queries.

```
SELECT count() FROM deltaLake(
  'https://clickhouse-public-datasets.s3.amazonaws.com/delta_lake/hits/')
  WHERE URL LIKE '%google%'

-- 4.396 sec.
```

Developers: Daniil Rubin, Ksenia Sumarokova, Flynn ucasfl. Since 22.11.

Integrations

Visualizations:
— official ClickHouse plugin for **Grafana**;
— official support for **Superset**;
— **HEX** and **Deepnote** support.

Data ingestion and processing:
— **Kafka Connect** integration;
— **Airflow**, **dbt** support.

Language drivers:
— official **Node.JS** driver;
— optimized **Go** driver;
— a new **Python** client.

Embedded Dashboards

https://play.clickhouse.com/dashboard

Operations

ClickHouse is easy to configure for your needs.

Dynamic Disks

Was:

```
storage_configuration:
  disks:
    web:
      type: web
      endpoint: 'https://clickhouse-public-datasets.s3.amazonaws.com/web/'
  policies:
    web:
      volumes:
        main:
          disk: web
```

```
ATTACH TABLE hits ...
ENGINE = MergeTree ORDER BY CounterID
SETTINGS storage_policy = 'web'
```

Dynamic Disks

Slightly better:

```
storage_configuration:
  disks:
    web:
      type: web
      endpoint: 'https://clickhouse-public-datasets.s3.amazonaws.com/web/'
```

```
ATTACH TABLE hits ...
ENGINE = MergeTree ORDER BY CounterID
SETTINGS disk = 'web'
```

No need for "storage policy" in simple cases.

Dynamic Disks

Much better:

```
ATTACH TABLE hits ...
ENGINE = MergeTree ORDER BY CounterID
SETTINGS disk = disk(
  type = 'web',
  endpoint = 'https://clickhouse-public-datasets.s3.amazonaws.com/web/')
```

100% dynamic configuration, no need to touch the configuration files.

Developers: Ksenia Sumarokova.

Composable Protocols

So, ClickHouse supports a lot of protocols:

— HTTP
— HTTPs
— Native TCP
— Native TCP wrapped in PROXYv1
— Native TCP with TLS
— MySQL (with TLS support)
— PostgreSQL (with TLS support)
— GRPC (with TLS)
— Replication protocol over HTTP
— Replication protocol over HTTPs
— Keeper client-server protocol;
— Keeper consensus protocol;
— ...

Composable Protocols

So, ClickHouse supports a lot of protocols.

How to configure all of them? What if:

— server has multiple network interfaces?
— enable one protocol on multiple ports?
— I want native TCP for localhost only and HTTPs from everywhere?
— I want different TLS certificates for different protocols?
— I want to wrap one protocol in another?

Developer: Yakov Olkhovskiy. Since 22.10.

```
<protocols>
  <tcp>
    <type>tcp</type>
    <host>::</host>
    <port>9000</port>
    <description>native protocol</description>
  </tcp>
  <tcp_secure>
    <type>tls</type>
    <impl>tcp</impl>
    <port>9440</port>
    <description>secure native protocol</description>
  </tcp_secure>
  <tcp_endpoint>
    <impl>tcp</impl>
    <host>0.0.0.0</host>
    <port>9001</port>
    <description>native protocol, another</description>
  </tcp_endpoint>
  <tcp_proxy>
    <type>proxy1</type>
    <impl>tcp</impl>
    <port>9100</port>
    <description>native protocol with PROXYv1</description>
  </tcp_proxy>
```

Q&A