My Favorite Features of 2023

— My favorite features

— Winter/Spring 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.

Parameterized Views

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

Requires access to the view and all the underlying tables.

Up next:
— Encapsulation of access control;
— Publishing views as API.

Developer: Smita Kulkarni.

Data Lakes

Now we support all three indistinguishable technologies:

Apache Hudi, **Apache Delta Lake**, **Apache Iceberg**

```
SELECT * FROM iceberg(
    'https://bucket.s3.amazonaws.com/test_table',
    s3 credentials...)

CREATE TABLE t ENGINE = Iceberg(
    'https://bucket.s3.amazonaws.com/test_table',
    s3 credentials...)

SELECT * FROM iceberg(named_conf
    'https://bucket.s3.amazonaws.com/test_table')
```

Developer: Kseniia Sumarokova, Daniil Rubin, ucasFL.

Data Lakes

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

— ClickHouse integrates with everything!
— s3, hdfs, https, mysql, postgres, sqlite, mongodb, jdbc, odbc...
— query the data without loading!

Advantages:
— these formats are somewhat resembling **MergeTree** allowing incremental data insertion, approaching to ClickHouse data formats;

Disadvantages:
— a band-aid solution to organize the data;

Data Lakes

Can we publish a MergeTree table into a data lake?

— Yes!

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.

Nested Dynamic Disks

```
ATTACH TABLE uk_price_paid UUID 'cf712b4f-2ca8-435c-ac23-c4393efe52f7'
(
    price UInt32,
    date Date,
    postcode1 LowCardinality(String),
    postcode2 LowCardinality(String),
    type Enum8('other' = 0, 'terraced' = 1, 'semi-detached' = 2, 'detach
    is_new UInt8,
    duration Enum8('unknown' = 0, 'freehold' = 1, 'leasehold' = 2),
    addr1 String,
    ...
)
ENGINE = MergeTree
ORDER BY (postcode1, postcode2, addr1, addr2)
SETTINGS disk = disk(
  type = cache,
  path = '/home/milovidov/.cache/clickhouse-local/',
  max_size = '10G',
  disk = disk(
    type = web,
    endpoint = 'https://raw.githubusercontent.com/ClickHouse/web-tables-
```

Developer: Kseniia Sumarokova.

Parallel Replicas

Without parallel replicas:

```
SELECT count() FROM github_events WHERE body ILIKE '%ClickHouse%';

Elapsed: 458.332 sec. 1.18 TB (2.58 GB/s.)
```

With 10 parallel replicas:

```
SET allow_experimental_parallel_reading_from_replicas = 1,
    max_parallel_replicas = 10;

SELECT count() FROM github_events WHERE body ILIKE '%ClickHouse%';

Elapsed: 40.284 sec. 1.18 TB (29.34 GB/s.)
```

With 100 parallel replicas:

```
Elapsed: 8.790 sec. 1.18 TB (134.44 GB/s.)
```

Developer: Nikita Mikhailov.

Parallel Replicas With Dynamic Shards

```
SET max_parallel_replicas = 100,
    parallel_replicas_custom_key = 'UserID',
    parallel_replicas_custom_key_filter_type = 'default';
 -- parallel_replicas_custom_key_filter_type = 'range'
```

Represents every **replica as a shard** for distributed queries
by pre-filtering data by remainder of division or a range.

Useful for large JOIN or GROUP BY with
the **distributed_group_by_no_merge** setting.

Developer: Antonio Andelic.

Parallel Replicas, Bonus

```
SELECT ... FROM s3(
    'https://bucket.s3.amazonaws.com/data*.parquet.zst')
```

```
SELECT ... FROM s3Cluster(
    cluster, 'https://bucket.s3.amazonaws.com/data*.parquet.zst')
```

Since version 23.5:

```
SELECT ... FROM url(
    'https://bucket.s3.amazonaws.com/data{01..99}.parquet.zst')
```

```
SELECT ... FROM urlCluster(
    cluster, 'https://bucket.s3.amazonaws.com/data{01..99}.parquet.zst')
```

Faster Parquet Reading

From S3 and URLs.

100 times faster.

Developer: Michael Kolupaev.

Autodetect Headers Of TSV/CSV

No need to choose between **CSV**, CSVWithNames, CSVWithNamesAndTypes for data import.

```
Was: SELECT * FROM file('data.csv', CSVWithNames);
Now: SELECT * FROM file('data.csv');
```

Just write **CSV**, and it will find the headers if there are any.

Everything detected automatically whenever possible!

Developer: Pavel Kruglov.

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 use_query_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.

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
```

Boring Features

Password Complexity Rules

```
$ cat /etc/clickhouse-server/config.d/rules.yaml

password_complexity:
    - rule:
        pattern: '.{12}'
        message: 'be at least 12 characters long'
    - rule:
        pattern: '\p{N}'
        message: contain at least 1 numeric character
    - rule:
        pattern: '\p{Lu}'
        message: contain at least 1 uppercase character
    - rule:
        pattern: '[^\p{L}\p{N}]'
        message: contain at least 1 special character
```

Developer: Nikolay Degterinsky.

Password Complexity Rules

```
:) CREATE USER vasyan
   IDENTIFIED WITH sha256_password BY 'qwerty123'

DB::Exception: Invalid password. The password should:
be at least 12 characters long,
contain at least 1 uppercase character,
contain at least 1 special character.
```

Developer: Nikolay Degterinsky.

Note: if clickhouse-client is being used,
the password will be checked and hashed on client side.

The server will never receive the plaintext password.

Credentials Cleansing

```
CREATE TABLE test AS mysql(
  'monty:3306', maria, table, 'videnius', 'qwerty123');

2022.12.15 07:51:10.997810 [ 2282939 ] {ea24d544-3e40-4f2a-9f0e-
<Debug> executeQuery: (from [::ffff:127.0.0.1]:47320)
CREATE TABLE test AS mysql('monty:3306', maria, table,
'videnius', '[HIDDEN]') (stage: Complete)

:) SHOW CREATE TABLE test

... AS mysql('monty:3306', maria, table, 'videnius', '[HIDDEN]')

:) SELECT query FROM system.query_log
   WHERE query LIKE 'CREATE TABLE test%'

CREATE TABLE test AS mysql(
  'monty:3306', maria, table, 'videnius', '[HIDDEN]')
```

More Boring Features

— **bcrypt** for password hashing;

— allow configuring the default password hashing;

```
CREATE USER test IDENTIFIED BY 'my_password';

config.d/password_type.yaml:

default_password_type: sha256_password
```

Not So Boring Features

Trailing Comma Before FROM

```
SELECT UserID,
       SearchPhrase,
       RegionID,
FROM test.hits
```

Was: Expected one of: token, Dot, OR, AND, BETWEEN, NOT BETWEEN, LIKE, ILIKE, NOT LIKE, NOT ILIKE, IN, NOT IN, GLOBAL IN, GLOBAL NOT IN, MOD, DIV, IS NULL, IS NOT NULL, alias, AS, Comma, FROM, PREWHERE, WHERE, GROUP BY, WITH, HAVING, WINDOW, ORDER BY, LIMIT, OFFSET, SETTINGS, UNION, EXCEPT, INTERSECT, INTO OUTFILE, FORMAT, end of query

Now: **Just works!**

Settings With `-` In Command Line

```
clickhouse-client --max_threads 1 --query "SELECT 1"
```

```
clickhouse-client --max-threads 1 --query "SELECT 1"
```

Was: Code: 552. DB::Exception: Unrecognized option '--max-threads'.

Now: **Just works!**

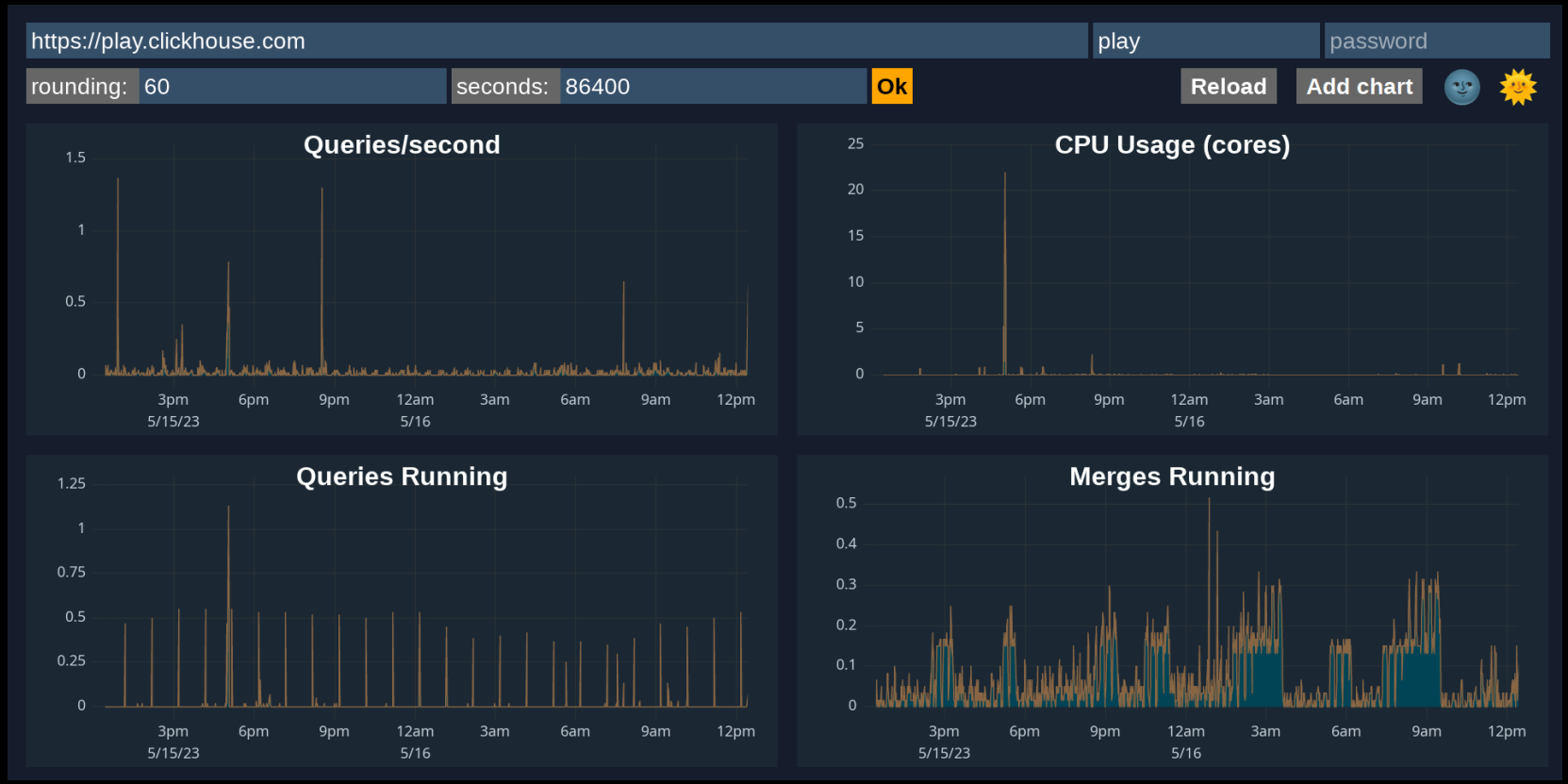
Bonus:

```
clickhouse-client —max-threads 1 --query "SELECT 1"
```

— It also works 👀

Embedded Dashboards

Embedded Dashboards



Query billions of rows in milliseconds

ClickHouse is the fastest and most resource efficient open-source database for real-time apps and analytics.

Deploy in 2 min | View documentation

Or download open-source ClickHouse ›

ClickHouse Cloud

— free 30-day trial with $300 credits up to 10 TB of data;

— affordable clusters for developers with full HA < $100/month

— now available in AWS and GCP!

Try it! https://clickhouse.cloud/.

Q&A