ClickHouse

What's new in ClickHouse 20.12 - 21.1

ClickHouse

#15111 Implement gRPC protocol in ClickHouse.

Vitaly Baranov - Yandex.

The implementation of gRPC protocol also supports compression, SSL,
getting progress and logs, authentication, parallel queries through the same channel,
cancellation of queries, sessions, external tables.

ClickHouse

/etc/clickhouse-server/config.xml

```
...
     <!-- <grpc_port>9100</grpc_port> -->
     <grpc>
     <enable_ssl>false</enable_ssl>

     <!-- The following two files are used only if enable_ssl=1 -->
     <ssl_cert_file>/path/to/ssl_cert_file</ssl_cert_file>
     <ssl_key_file>/path/to/ssl_key_file</ssl_key_file>
...
```

ClickHouse

example (python) :
https://github.com/ClickHouse/ClickHouse/blob/master/utils/grpc-client/clickhouse-grpc-client.py

```
utils/grpc-client/clickhouse-grpc-client.py -q "SELECT sum(number) FROM numbers(10)"

cat a.txt | utils/grpc-client/clickhouse-grpc-client.py -q "INSERT INTO temp FORMAT TSV"
```

ClickHouse

#17144 *.zst compression/decompression support for data import and export.
It enables using *.zst in file() function and Content-encoding: zstd in HTTP client.

Abi Palagashvili

#8526 support for brotli (br) compression in file-related storages and table functions.

alexey-milovidov - Yandex

ClickHouse

```
CREATE TABLE file (x UInt64) ENGINE = File(TSV, 'data1.tsv.br');


SELECT count(), max(x)
FROM file('data{1,2}.tsv.{gz,br}', TSV, 'x UInt64');



$ echo "SELECT 1" | brotli | curl -sS --data-binary @- \
   -H 'Content-Encoding: br' "${CLICKHOUSE_URL}"

$ echo "SELECT 1" | zstd -c | curl -sS --data-binary @- \
   -H 'Content-Encoding: zstd' "${CLICKHOUSE_URL}";
```

Supported Content-Encoding: gzip, deflate, br, xz, zstd

ClickHouse

#15806 new data type Map.
First version for Map only supports String type of key and value. ().

hexiaoting

```
SET allow_experimental_map_type = 1;


create table table_map (a Map(String, String)) engine = Memory;

insert into table_map values
  ({'name':'zhangsan', 'gender':'male'}),
  ({'name':'lisi', 'gender':'female'});


SELECT a['name'] FROM table_map;

┌─arrayElement(a, 'name')─┐
│ zhangsan                │
│ lisi                    │
└─────────────────────────┘
```

ClickHouse

#16338 Implement UNION DISTINCT
and treat the plain UNION clause as UNION DISTINCT by default.

Add a setting union_default_mode that allows to treat it as
UNION ALL or require explicit mode specification. (DISTINCT / ALL)

flynn

```
SET union_default_mode = 'DISTINCT';

SELECT 'a' UNION SELECT 'a';
┌─'a'─┐
│ a   │
└─────┘
```

```
SET union_default_mode = 'ALL';

SELECT 'a' UNION SELECT 'a';
┌─'a'─┐
│ a   │
└─────┘
┌─'a'─┐
│ a   │
└─────┘
```

#11617 Parallel formatting for data export.

Nikita Mikhaylov Yandex

```
time clickhouse-client
-q 'select number, number/11111, toString(number) from numbers(100000000) format TSV' > /dev/null

      real    0m3.594s
      user    0m10.867s
      sys     0m0.308s


time clickhouse-client --output_format_parallel_formatting=0
-q 'select number, number/11111, toString(number) from numbers(100000000) format TSV' > /dev/null

      real    0m9.410s
      user    0m9.128s
      sys     0m0.276s
```

ClickHouse

#7649 Added functions for calculation of minHash and simHash of text n-grams and shingles.

They are intended for semi-duplicate search.

Simhash, Minhash, bitHammingDistance, tupleHammingDistance

flynn

```
SELECT ngramSimHash('what a cute cat.');
SELECT ngramSimHashCaseInsensitive('what a cute cat.');
SELECT ngramSimHashUTF8('what a cute cat.');
SELECT ngramSimHashCaseInsensitiveUTF8('what a cute cat.');

SELECT wordShingleSimHash('what a cute cat.');
SELECT wordShingleSimHashCaseInsensitive('what a cute cat.');
SELECT wordShingleSimHashUTF8('what a cute cat.');
SELECT wordShingleSimHashCaseInsensitiveUTF8('what a cute cat.');

SELECT ngramMinHash('what a cute cat.');
SELECT ngramMinHashCaseInsensitive('what a cute cat.');
SELECT ngramMinHashUTF8('what a cute cat.');
SELECT ngramMinHashCaseInsensitiveUTF8('what a cute cat.');

SELECT wordShingleMinHash('what a cute cat.');
SELECT wordShingleMinHashCaseInsensitive('what a cute cat.');
SELECT wordShingleMinHashUTF8('what a cute cat.');
SELECT wordShingleMinHashCaseInsensitiveUTF8('what a cute cat.');

SELECT tupleHammingDistance((1, 2), (3, 4));
SELECT bitHammingDistance(100, 100000);
```

```
select ngramSimHash('Universal Avenue') a, ngramSimHash('Universe Avenue') b, a/b diff

┌──────────a─┬──────────b─┬─────────────────diff─┐
│ 1577599152 │ 1580943536 │ 0.997884564550318 │
└────────────┴────────────┴─────────────────────┘
```

```
select ngramSimHash('Purple flowers') a, ngramSimHash('Green grass') b, a/b diff

┌──────────a─┬──────────b─┬─────────────────diff─┐
│ 1485423320 │ 3599596288 │ 0.4126638659318453 │
└────────────┴────────────┴─────────────────────┘
```

ClickHouse

#16883 Added mannWitneyUTest, studentTTest and welchTTest aggregate functions. Refactored rankCorr a bit.

Nikita Mikhaylov - Yandex

ClickHouse

```
SELECT rankCorr(number, number - 33)
FROM numbers(10)

┌─rankCorr(number, minus(number, 33))─┐
│                                   1 │
└─────────────────────────────────────┘
```

```
SELECT rankCorr(number, 1 / number)
FROM numbers(100)

┌─rankCorr(number, divide(1, number))─┐
│                                  -1 │
└─────────────────────────────────────┘
```

ClickHouse

#13403 ALTER UPDATE/DELETE IN PARTITION
with partition pruning in ReplicatedMergeTree

Vladimir Chebotarev - Altinity.

ClickHouse

```
CREATE TABLE T (p Int64, d Date, s String)
ENGINE=ReplicatedMergeTree('/clickhouse/T', '{replica}')
PARTITION BY toYYYYMM(d) ORDER BY p;

INSERT INTO T VALUES(1, '2020-01-01', 'x'), (2, '2020-10-01', 'y');

SELECT *, _part FROM T;
┌─p─┬──────────d─┬─s─┬─_part─────┐
│ 1 │ 2020-01-01 │ x │ 202001_0_0_0 │
└───┴────────────┴───┴──────────┘

┌─p─┬──────────d─┬─s─┬─_part─────┐
│ 2 │ 2020-10-01 │ y │ 202010_0_0_0 │
└───┴────────────┴───┴──────────┘
```

ClickHouse

```
ALTER TABLE T UPDATE s = 'z' WHERE p = 1 and d = '2020-01-01';

SELECT *, _part FROM T;
```

p	d	s	_part
1	2020-01-01	z	202001_0_0_0_1

p	d	s	_part
2	2020-10-01	y	202010_0_0_0_1

ClickHouse

```
ALTER TABLE T  UPDATE s = 'z'
   IN PARTITION tuple(toYYYYMM(toDate('2020-01-01')))
   WHERE p = 1;

SELECT *, _part FROM T;
```

```
┌─p─┬──────────d─┬─s─┬─_part──────┐
│ 1 │ 2020-01-01 │ z │ 202001_0_0_0_1 │
└───┴────────────┴───┴────────────┘

┌─p─┬──────────d─┬─s─┬─_part─────┐
│ 2 │ 2020-10-01 │ y │ 202010_0_0_0 │
└───┴────────────┴───┴──────────┘
```

ClickHouse

#17642 DETACH TABLE/VIEW ... PERMANENTLY

filimonov - Altinity.

so that after restarting the table does not reappear back automatically
on restart (only by explicit request).
The table can still be attached back using the short syntax ATTACH TABLE.

ClickHouse

```
DETACH table T PERMANENTLY;

service clickhouse-server restart

select * from T
DB::Exception: Table dw.T doesn't exist.

attach table T;

select * from T;
```

p	d	s
1	2020-01-01	x

#17846 Extended OPTIMIZE ... DEDUPLICATE
syntax to allow explicit list of columns to check for duplicates on.

Vasily Nemkov - Altinity.

ClickHouse

```
CREATE TABLE T(country String, city String, d Date,
               some_col1 Int64, some_col2 Int64)
Engine=MergeTree Order by (country, city);

INSERT INTO T VALUES('FR', 'Leon', '2020-01-01', 33, 42);
INSERT INTO T VALUES('FR', 'Leon', '2020-01-01', 32, 41);
INSERT INTO T VALUES('FR', 'Leon', '2020-01-01', 35, 41);

OPTIMIZE TABLE T DEDUPLICATE
   BY * EXCEPT (some_col1, some_col2);

SELECT * FROM T

┌─country─┬─city─┬──────────d─┬─some_col1─┬─some_col2─┐
│ FR      │ Leon │ 2020-01-01 │        33 │        42 │
└─────────┴──────┴────────────┴───────────┴───────────┘
```

ClickHouse

```
OPTIMIZE TABLE table DEDUPLICATE; -- the old one

OPTIMIZE TABLE table DEDUPLICATE BY *; -- excludes MATERIALIZED columns

OPTIMIZE TABLE table DEDUPLICATE BY * EXCEPT colX;

OPTIMIZE TABLE table DEDUPLICATE BY * EXCEPT (colX, colY);

OPTIMIZE TABLE table DEDUPLICATE BY col1,col2,col3;

OPTIMIZE TABLE table DEDUPLICATE BY COLUMNS('column-matched-by-regex');

OPTIMIZE TABLE table DEDUPLICATE BY COLUMNS('column-matched-by-regex') EXCEPT colX;

OPTIMIZE TABLE table DEDUPLICATE BY COLUMNS('column-matched-by-regex') EXCEPT (colX, colY);
```

ClickHouse

#15511 ALTER TABLE ... DROP PART 'part_name'

nvartolomei - Cloudflare

ClickHouse

```
CREATE TABLE T (p Int64, d Date, s String)
ENGINE=ReplicatedMergeTree('/clickhouse/T', '{replica}')
PARTITION BY toYYYYMM(d) ORDER BY p;

INSERT INTO T VALUES (1, '2020-01-01', 'x');
INSERT INTO T VALUES (2, '2020-01-02', 'y');
```

ClickHouse

```
SELECT *, _part FROM T;

┌─p─┬──────────d─┬─s─┬─_part─────────┐
│ 2 │ 2020-01-02 │ y │ 202001_1_1_0 │
└───┴────────────┴───┴───────────────┘

┌─p─┬──────────d─┬─s─┬─_part─────────┐
│ 1 │ 2020-01-01 │ x │ 202001_0_0_0 │
└───┴────────────┴───┴───────────────┘
```

```
ALTER TABLE T DROP PART '202001_0_0_0';
```

```
SELECT *, _part FROM T;

┌─p─┬──────────d─┬─s─┬─_part─────────┐
│ 2 │ 2020-01-02 │ y │ 202001_1_1_0 │
└───┴────────────┴───┴───────────────┘
```

#16895 Remove empty parts after they were pruned by TTL, mutation, or collapsing merge algorithm.

Anton Popov - Yandex

#15073 Add EmbeddedRocksDB table engine (can be used for dictionaries).

sundyli - ByteDance

ClickHouse

```
CREATE TABLE T1 ( P Int64, S String )
ENGINE = EmbeddedRocksDB
PRIMARY KEY P;

INSERT INTO T1 VALUES (1, '');

INSERT INTO T1 VALUES (1, 'updated');


SELECT * FROM T1;

┌─P─┬─S───────┐
│ 1 │ updated │
└───┴─────────┘
```

ClickHouse

```
INSERT INTO T1 select number, toString(number)
from numbers(10 000 000);


select * from T1 where P = 99999;
┌─────P─┬─S─────┐
│ 99999 │ 99999 │
└───────┴───────┘
1 rows in set. Elapsed: 0.001 sec.


INSERT INTO T1 VALUES (99999, 'updated');
1 rows in set. Elapsed: 0.001 sec.

select * from T1 where P = 99999;
┌─────P─┬─S───────┐
│ 99999 │ updated │
└───────┴─────────┘
1 rows in set. Elapsed: 0.001 sec.
```

#16123 Add setting aggregate_functions_null_for_empty for SQL standard compatibility. This option will rewrite all aggregate functions in a query, adding -OrNull suffix to them.

flynn - Institute of Computing Technology, Beijing

ClickHouse

```
CREATE TABLE T (p Int64, d Date, s String)
ENGINE=MergeTree
PARTITION BY toYYYYMM(d) ORDER BY p;

INSERT INTO T VALUES (1, '2020-01-01', 'x'), (2, '2020-10-01', 'y');
```

```
SELECT max(p) FROM T WHERE s = '';
```

```
┌─max(p)─┐
│      0 │
└────────┘
```

```
SELECT max(p) FROM T WHERE s = ''
SETTINGS aggregate_functions_null_for_empty = 1;
```

```
┌─maxOrNull(p)─┐
│         null │
└──────────────┘
```

ClickHouse

#16575 Subqueries in WITH section (CTE) can reference previous subqueries in WITH section by their name.

Amos Bird

```
WITH
T1 as (select number as c1 from numbers(10000) where number%2),
T2 as (select max(c1) sm from T1),
T3 as (select * from T1 where c1 = (select sm from T2))
select * from T3

┌───c1─┐
│ 9999 │
└──────┘
```

#16253 Now we can safely prune partitions with exact match.

Useful case: Suppose table is partitioned by intHash64(x) % 100 and the query has condition on on intHash64(x) % 100 verbatim, not on x.

Amos Bird

```
CREATE TABLE T (p Int64, d Date, tenant_id Int64, s String)
ENGINE=MergeTree
PARTITION BY (toYYYYMM(d), tenant_id % 100)
ORDER BY p;
```

```
SELECT count() FROM T WHERE (tenant_id = 42) AND (d = '1970-01-11')
┌─count()─┐
│    97   │
└─────────┘
1 rows in set. Elapsed: 0.003 sec. Processed 30.10 thousand rows
```

```
select count() from T where identity(tenant_id) = 42 and d = '1970-01-11';
┌─count()─┐
│    97   │
└─────────┘
1 rows in set. Elapsed: 0.010 sec. Processed 3.01 million rows
```

ClickHouse

```
CREATE TABLE T (p Int64, d Date, tenant_id Int64,
     part_id materialized tenant_id % 100,
     s String)
ENGINE=MergeTree
PARTITION BY (toYYYYMM(d), part_id)
ORDER BY p;
```

```
select count() from T where tenant_id = 42 and part_id = 42%100 and d = '1970-01-11';
┌─count()─┐
│    97   │
└─────────┘
1 rows in set. Elapsed: 0.014 sec. Processed 30.10 thousand rows
```

ClickHouse

#18637 SimpleAggregateFunction in SummingMergeTree.
Now it works like AggregateFunction.

Amos Bird

ClickHouse

```
CREATE TABLE T
(
      country String,
      city String,
      some_sum Int64,
      some_max SimpleAggregateFunction(max, Int32),
      some_gr SimpleAggregateFunction(groupUniqArrayArray, Array(String))
)
ENGINE = SummingMergeTree
ORDER BY (country, city);

INSERT INTO T VALUES ('FR', 'Leon', 1, 1, ['j']);
INSERT INTO T VALUES ('FR', 'Leon', 1, 10, ['k']);

OPTIMIZE TABLE T FINAL;
```

country	city	some_sum	some_max	some_gr
FR	Leon	2	10	['k','j']