Introduction to Clickhouse

Ilia Alshanetsky

@iliaa - ilia@ilia.ws

Me, myself and I ;-)

- CTO @ Silofit – We are Hiring!!
- PHP Core Contributor & Ex-Release Master
- Author & Co-Author of multiple PHP extensions
- Security Nerd, wrote Guide to PHP Security
- Fascinated by making things faster
- Occasional Photographer

What is Clickhouse?

Columnar Data Warehouse

Real-Time Analytics Engine

Supports SQL Like Syntax

FAST !!!!

Shared Nothing Architecture

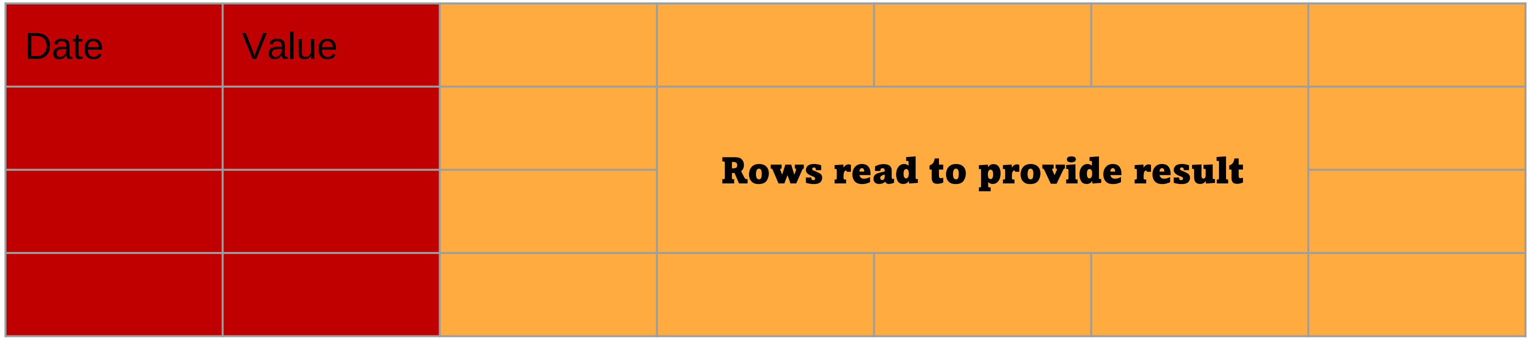
Parallel & Vectorized Execution

Highly Efficient Storage

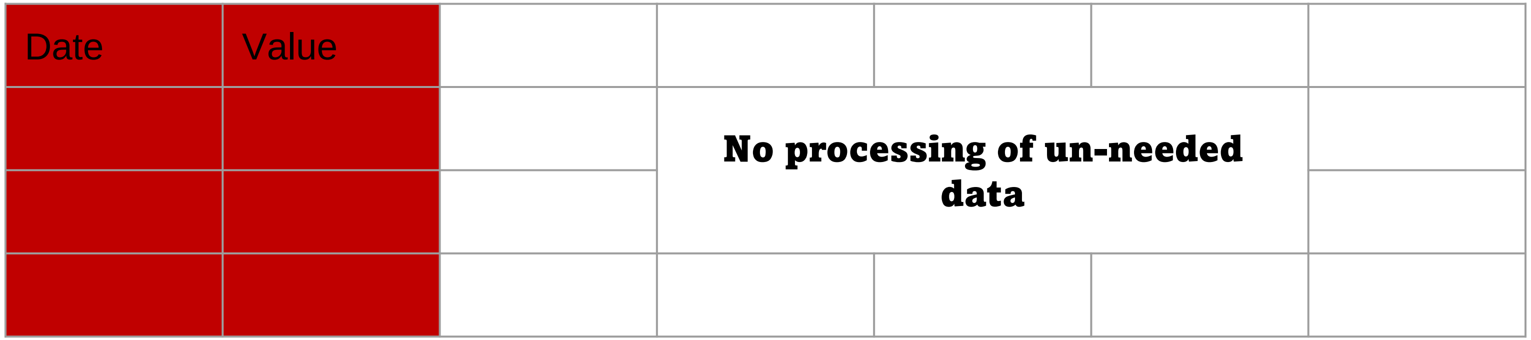
Open Source – Apache 2.0

Columnar?!

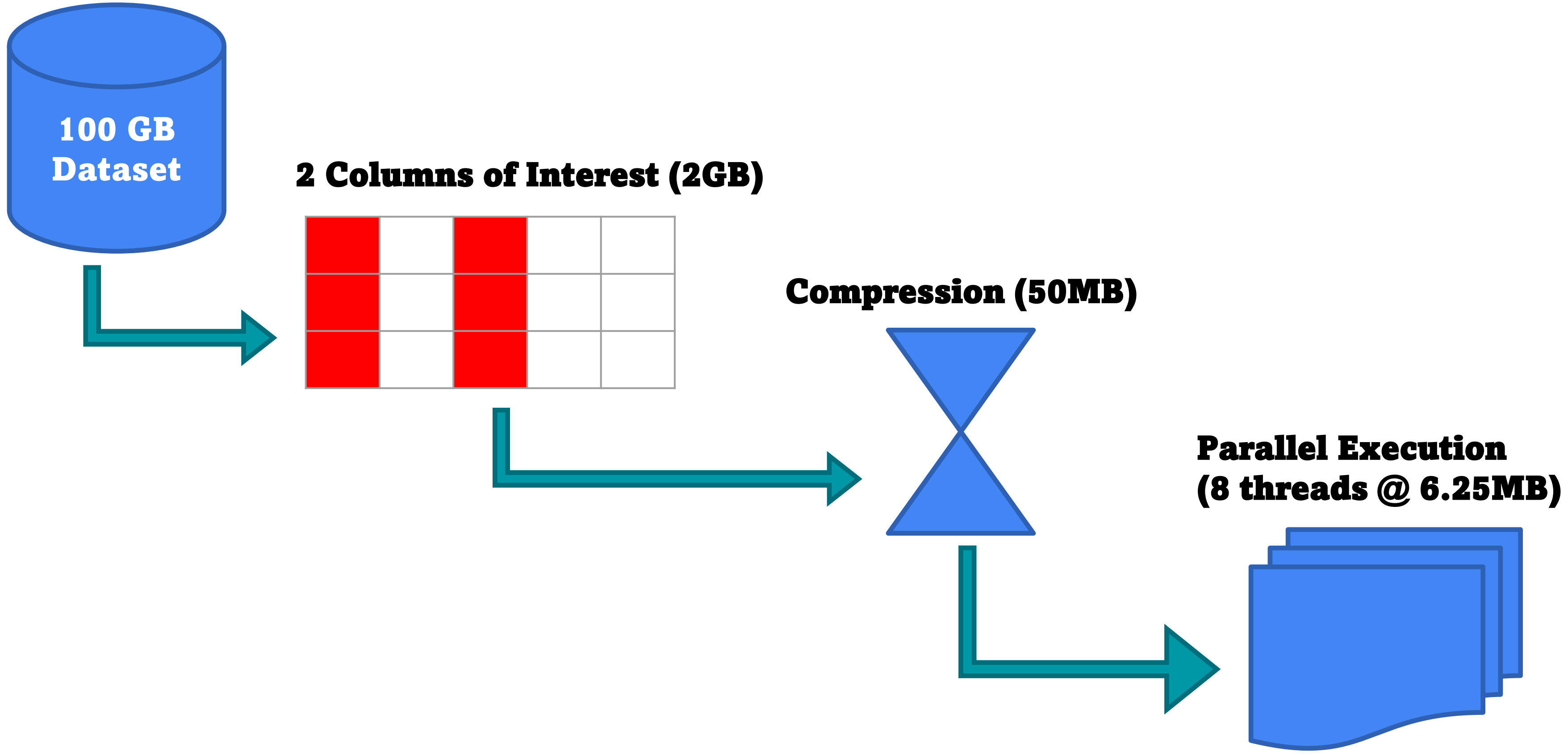
Traditional SQL Database - Row Based



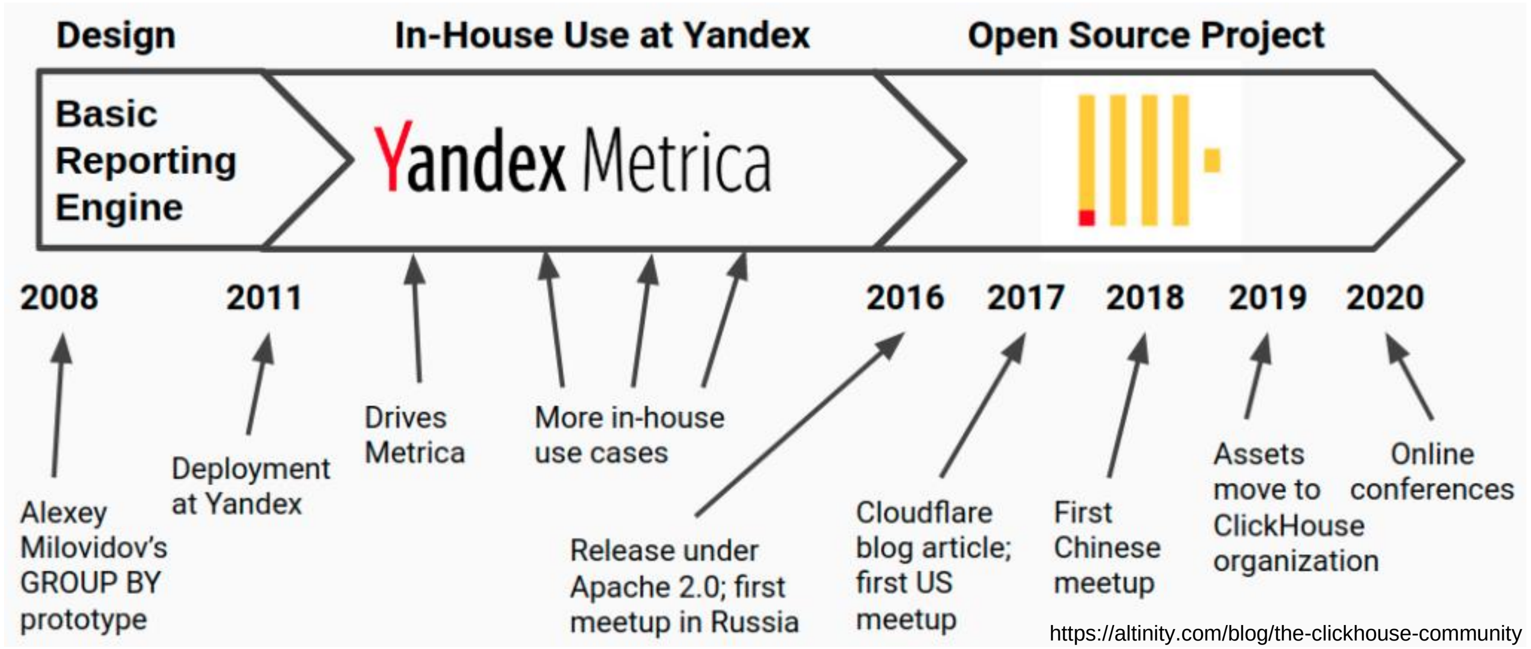
Clickhouse - Column Based



Less is ~~more~~ Faster!


100 GB Dataset
2 Columns of Interest (2GB)
Compression (50MB)
Parallel Execution (8 threads @ 6.25MB)

Bona Fide & History


Design
In-House Use at Yandex
Open Source Project
Basic Reporting Engine
Yandex Metrica
2008
2011
2016
2017
2018
2019
2020
Alexey Milovidov's GROUP BY prototype
Deployment at Yandex
Drives Metrica
More in-house use cases
Release under Apache 2.0; first meetup in Russia
Cloudflare blog article; first US meetup
First Chinese meetup
Assets move to ClickHouse organization
Online conferences
https://altinity.com/blog/the-clickhouse-community

Today, in-use by 1000s of Companies all over the world!

Docker Installation

```
mkdir $HOME/ch-data

docker run -d --name clickhouse-server \
 --ulimit nofile=262144:262144 \
 --volume=$HOME/ch-data:/var/lib/clickhouse \
 -p 8123:8123 -p 9000:9000 \
 yandex/clickhouse-server
```

Increase file limit

Persist Data

Expose Interface Ports (http + native)

Connecting to Clickhouse

```
clickhouse-client \
  --host=localhost \
  --port=9000 \
  --secure --user=clickhouse --password=secret
  -d my_database
```

All you need locally

Optional
Networking

Optional
Security & Authentication

Optional
Pick a Database

There are a few hundred options on CLI beyond that ;-)

Create & Select a Database

```
$ clickhouse-client

ClickHouse client version 18.16.1.

Connecting to localhost:9000.

Connected to ClickHouse server version 21.9.4 revision 54449.

:) create database letsgo

CREATE DATABASE letsgo

Ok.

0 rows in set. Elapsed: 0.017 sec.
```

To change from `default` **database (yes, that is its name)**

```
:) use letsgo

USE letsgo

Ok.

0 rows in set. Elapsed: 0.004 sec.
```

Our First Table

```
CREATE TABLE IF NOT EXISTS actions (
 device_id UInt32,
 user UUID,
 dt Date DEFAULT toDate(ts),
 ts DateTime('Toronto/Canada'),
 action UInt8,
 data String
) ENGINE = MergeTree()
PARTITION BY toYYYYMM(dt)
ORDER BY (device_id, ts)
```

Auto-resolve date from timestamp

Stored as timestamp, time-zone controls the parsing & output

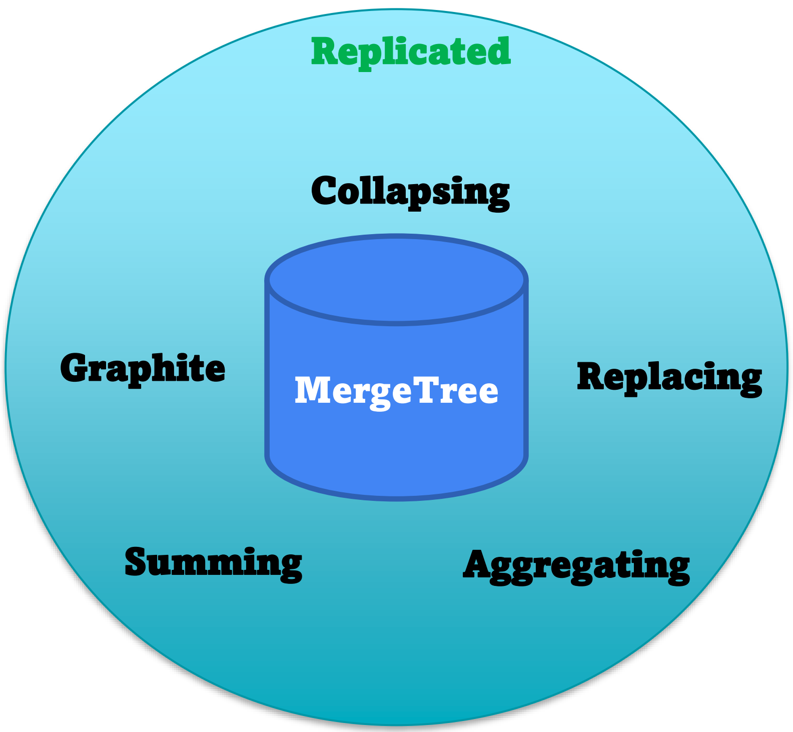
No "char" or "text", just string!

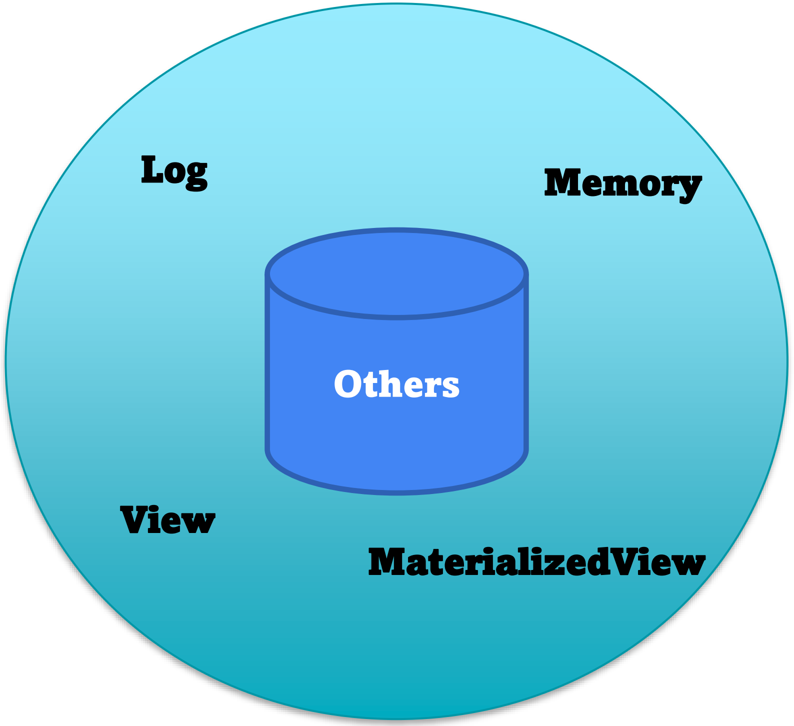
Storage Engine

Partitioning

Index & Sort

Engine Families





```
ie. ReplicatedCollapsingMergeTree()
```

Boring Data Load

```
INSERT INTO letsgo.actions (device_id, user, ts, action)

VALUES

(1, ‘e0147c53-fb62-4bdc-aa5b-408b2f2a6048’, ‘2022-01-01 13:12:15’, 10),

(2, ‘e0147c53-fb62-4bdc-aa5b-408b2f2a6048’, 1645121790, 11);
```

```
INSERT INTO letsgo.actions (device_id, user, ts, action) VALUES

Ok.

2 rows in set. Elapsed: 0.014 sec.
```

Think Different!

```
device_id,user,ts,action
“1”,”e0147c53-fb62-4bdc-aa5b-408b2f2a6048”,”2022-01-01 13:12:15”,”10”

“2”,”e0147c53-fb62-4bdc-aa5b-408b2f2a6048”,”2022-01-01 13:12:16”,”11”
```

```
cat mydata.csv | clickhouse-client \

--query='INSERT INTO letsgo.actions FORMAT CSVWithNames’
```

```
CDATA='INSERT%20INTO%20letsgo.actions%20Format%20CSVWithNames’ \

curl -v "http://localhost:8123/?query=${CDATA}" --data-binary @mydata.csv
```

Let’s Get Really Creative!

```
:) select * from system.formats where is_input = 1;
```

30+

JSON	**Multiple JSON formats are supports**
TabSeparated	**If commas are not your style**
MsgPack	**Efficient binary serialization format**
Protobuf	**Google Protocol Buffers**
Avro	**Apache Avro and Confluent/Kafka**
Regexp	**Load data captured by regex**

Peek behind the curtain of MergeTree

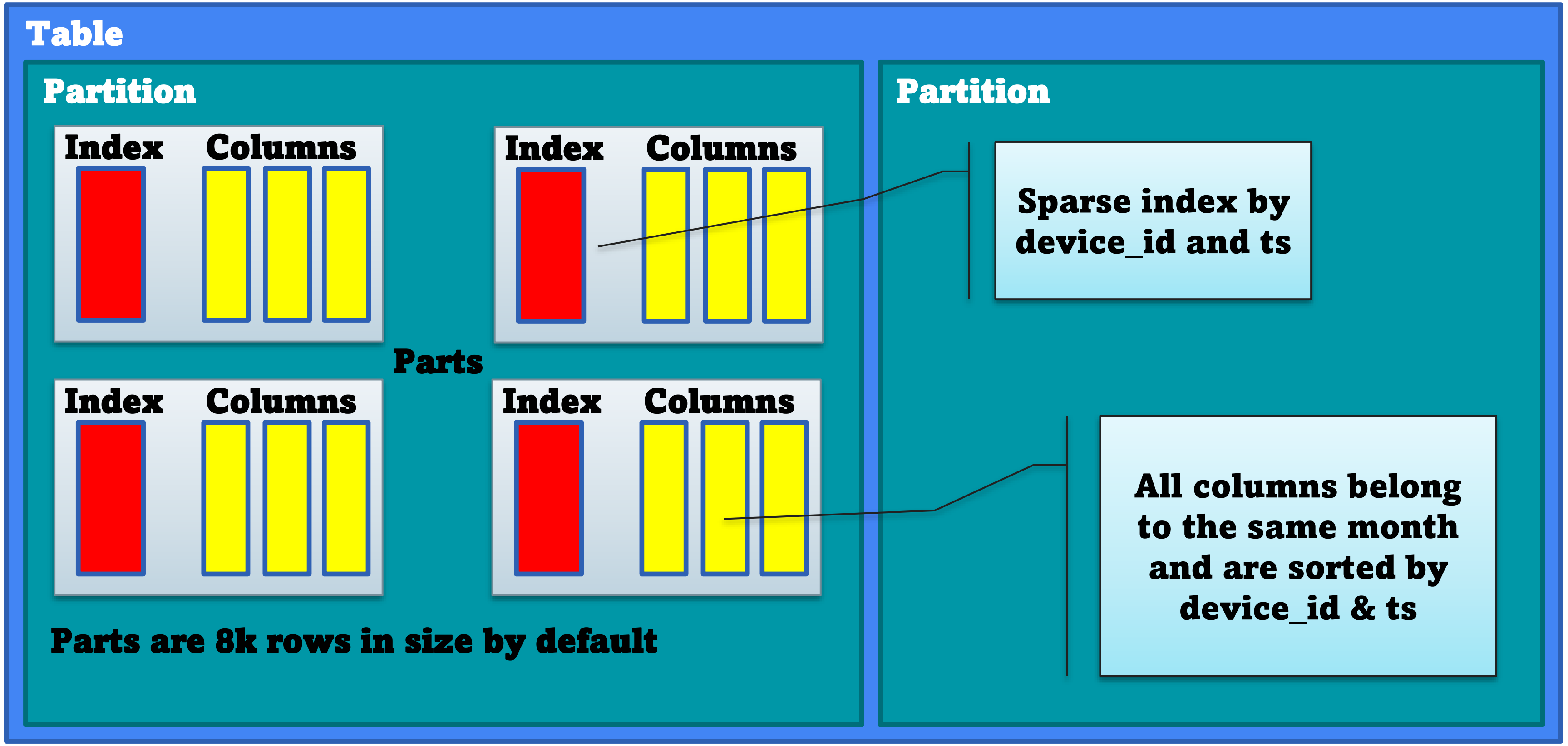
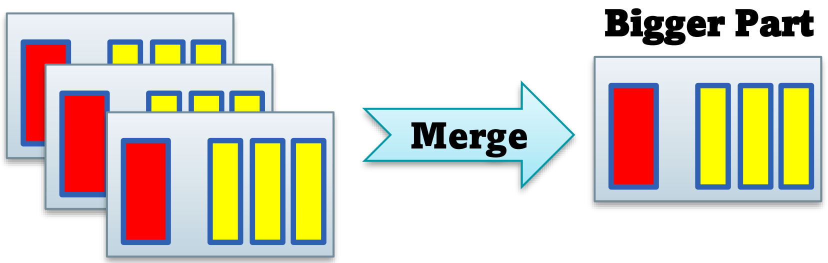


Table
Partition
Index
Columns
Index
Columns
Parts
Index
Columns
Index
Columns
Parts are 8k rows in size by default
Partition
Sparse index by device_id and ts
All columns belong to the same month and are sorted by device_id & ts

One Part at a Time

```
:) select path from system.parts where table = 'trace_log’;

┌─path─────────────────────────────────────────────────────────────────────┐
│ /var/lib/clickhouse/store/969/969fd8de-eb3c-436b-969f-d8deeb3ce36b/202202_1_70_16/ │
│ /var/lib/clickhouse/store/969/969fd8de-eb3c-436b-969f-d8deeb3ce36b/202202_1_72_17/ │
│ /var/lib/clickhouse/store/969/969fd8de-eb3c-436b-969f-d8deeb3ce36b/202202_71_71_0/ │
│ /var/lib/clickhouse/store/969/969fd8de-eb3c-436b-969f-d8deeb3ce36b/202202_72_72_0/ │
└──────────────────────────────────────────────────────────────────────────┘

↘ Progress: 4.00 rows, 436.00 B (1.02 thousand rows/s., 110.77 KB/s.)
4 rows in set. Elapsed: 0.004 sec.
```

```
:) select partition, path from system.parts where table = 'trace_log’;

┌─partition─┬─path───────────────────────────────────────────────────────────────┐
│ 202202    │ /var/lib/clickhouse/store/969/969fd8de-eb3c-436b-969f-d8deeb3ce36b/202202_1_72_17/ │
└───────────┴────────────────────────────────────────────────────────────────────┘

↖ Progress: 1.00 rows, 124.00 B (136.83 rows/s., 16.97 KB/s.)
1 rows in set. Elapsed: 0.008 sec.
```



Doing “Stuff” aka Selecting…

```
SELECT action,
 toMonth(dt) AS Month,
 count() AS actions
FROM letsgo.actions
WHERE dt >= 2022 AND action = 10
GROUP BY Month, action
HAVING actions > 1000
ORDER BY actions DESC
LIMIT 100
```

Lots of date slicing functions

No need for 1 or * inside count

Largely looks like traditional SQL

Updating & Deleting

```
ALTER TABLE mydata

UPDATE value = 2

WHERE parameter = 3;
```

This is expensive!

```
ALTER TABLE mydata

DELETE WHERE param = 7;
```

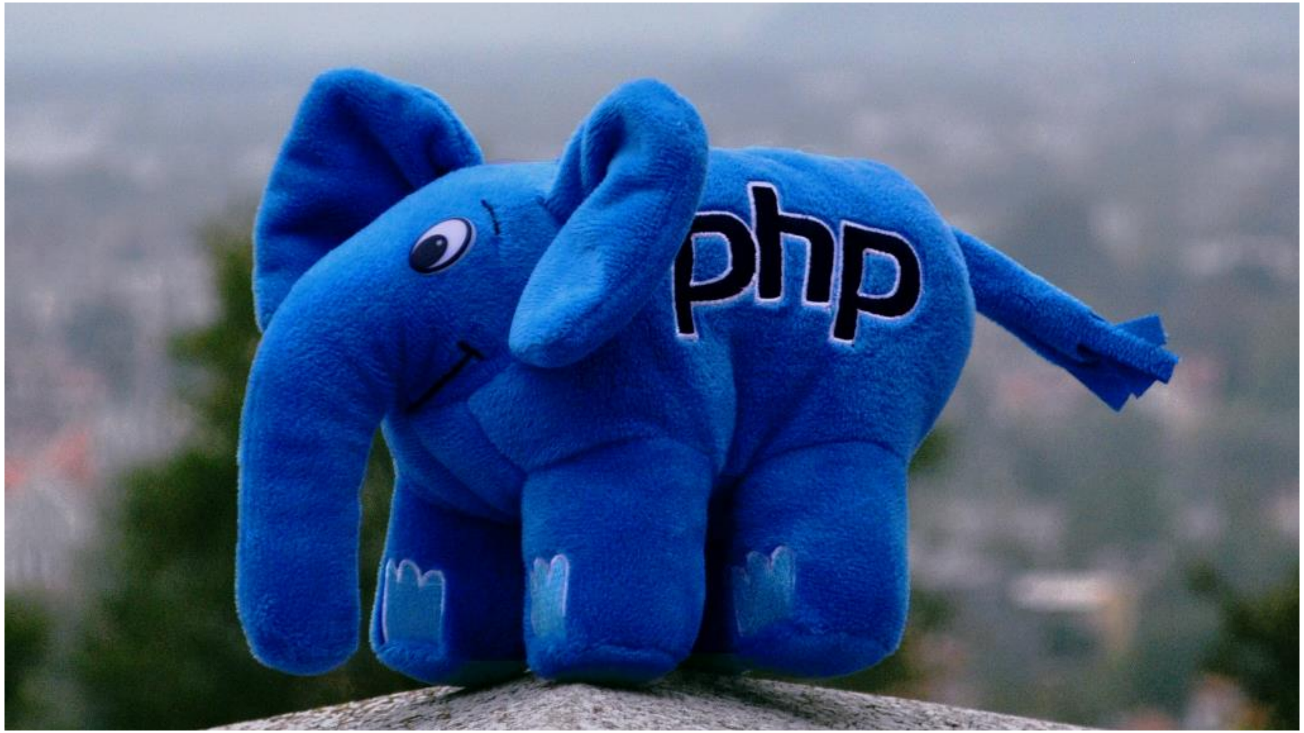
Can be slow enough, there is a way to identify progress…

```
:) SELECT command, is_done FROM system.mutations WHERE table = 'mydata'
```

Handy Utility Commands

SHOW DATABASES	**Show list of available databases**
SHOW TABLES *[from database]*	**Show list of tables, optionally in provided database**
DESC TABLE *table-name*	**Show structure of a table**
SELECT * FROM system.settings **FORMAT Vertical**	**Show all settings & toggle vertical output format**
SELECT query_id, query, elapsed FROM system.processes	**Show running queries**
KILL QUERY WHERE query_id = '*<id>*'	**Kill running query**
KILL MUTATION WHERE mutation_id = ‘<id>'	**Kill running mutation (alter command)**

PHP Interface



You got Options!

Native PHP	PHP Extension
Easy to Install	FAST!
All Versions Supported	Native Binary Protocol
HTTP Protocol	Convenience Interface Methods
Allows Async Queries (Curl)	Protocol Level Compression
https://github.com/smi2/phpClickHouse	https://github.com/SeasX/SeasClick https://github.com/iliaal/SeasClick

Going Native - Initialization

```
composer require smi2/phpclickhouse
```

```php
<?php
$config = [
    'host' => '192.168.1.1',
    'port' => '8123',
    'username' => 'default',
    'password' => ''
];
$db = new ClickHouseDB\Client($config);
$db->database('default');
$db->setTimeout(1.5);      // 1500 milliseconds
$db->setConnectTimeOut(5); // 5 seconds
```

```php
$db->write('
    CREATE TABLE IF NOT EXISTS mydata (
        ts DateTime,
        foo Int32,
        bar String
    ) ENGINE = MergeTree() PARTITION BY toYYYYMM(ts) ORDER BY (foo, ts)
');
```

Going Native – Writing & Reading

```php
$db->insert('mydata',
    [
        [time(), 1, 'hello'],
        [time(), 2, 'world'],
        [time(), 3, 'click'],
        [time(), 4, 'house'],
    ],
    ['ts', 'foo', 'bar']
);
```

```php
$statement = $db->select('SELECT * FROM mydata LIMIT 2');

// get a count of returned rows
$statement->count();

// fetch 1st row
$statement->fetchOne();

// fetch all results
$statement->rows();

// get timing information
$statement->totalTimeRequest();
```

```php
$query1 = $db->selectAsync('SELECT 1 as ping');
$query2 = $db->selectAsync('SELECT 2 as ping');

// execute queries
$db->executeAsync();

// process ready results
$query1->rows();
$query2->fetchOne('ping');
```

Need for Speed!

```
git clone https://github.com/SeasX/SeasClick.git
cd SeasClick
phpize
./configure
make && make install
```

```php
$config = [
    "host" => "localhost",
    "port" => 9000,
    "compression" => true
];
$client = new SeasClick($config);
```

```php
$client->execute('
    CREATE TABLE IF NOT EXISTS mydata (
        ts DateTime,
        foo Int32,
        bar String
    ) ENGINE = MergeTree() PARTITION BY toYYYYMM(ts) ORDER BY (foo, ts)
');
```

PHP Extension – Writing & Reading

```php
$client->insert('mydata',
    ['ts', 'foo', 'bar'],
    [
        [time(), 1, 'hello'],
        [time(), 2, 'world'],
        [time(), 3, 'click'],
        [time(), 4, 'house'],
    ],
);
```

```php
$result = $client->select("SELECT * FROM mydata LIMIT 2"); // returns array of arrays

// Fetch 1st value of 1st column, formatting date into YYYY-MM-DD HH:MM:SS format
$client->select("SELECT ts FROM mydata", [], SeasClick::FETCH_ONE|SeasClick::DATE_AS_STRINGS);

// Fetch column values as an array of values
$client->select("SELECT ts FROM mydata", [], SeasClick::FETCH_COLUMN);

// Return as associated array keyed foo [foo => bar]
$client->select("SELECT foo, bar FROM mydata", [], SeasClick::FETCH_KEY_PAIR);
```

Clickhouse Resources

Official Clickhouse Documentation - https://clickhouse.com/docs/en/

Lots of Great Presentations & Guides on Clickhouse - https://altinity.com/resources/

PHP Interface Library - https://github.com/smi2/phpClickHouse

Native PHP Extension
https://github.com/SeasX/SeasClick
https://github.com/iliaal/SeasClick

Clickhouse Slack Channel - http://clickhousedb.slack.com/

Thank you for listening!