chDB

Blazing Fast SQL Engine for Data Science

July, 2024 @Prague

ClickHouse

★ Auxten: about me

👨🏻‍💻 Experience in RecSys, Database

- Technical Director of ClickHouse core team
- Principal Engineer in Shopee (DB for RecSys)

❤️ Love Open Source!

- Contributed to ClickHouse, Jemalloc, K8s, Memcached, CockroachDB, Superset
- Creator of chDB, CovenantSQL

- auxten.com

What is chDB?

in-process

SQL

OLAP Engine

powered by ClickHouse

PostgreSQL ? but no Server

Python dict ? with SQL support

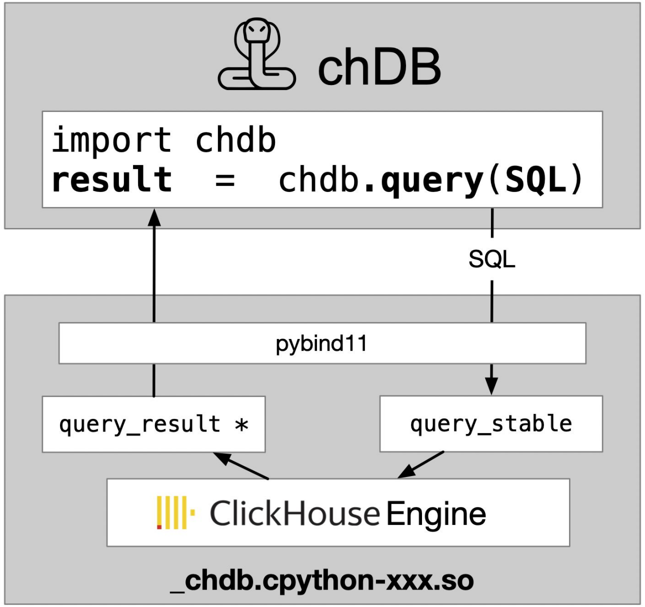
SQLite ? but Columnar

ClickHouse + python™

Rocket Engine on a Bicycle

- ★ **chDB** 🚴 + 🚀
  - ○ In-process SQL OLAP Engine, powered by ClickHouse
  - ○ Serverless. No need to install/run ClickHouse
  - ○ Supports all ClickHouse Functions & Formats *(24.5)*
  - ○ Support for Python DB API 2.0 and Dataframes
  - ○ Support for Stateful Query Sessions w/ Autoclean
  - ○ Minimized data copy from C++ to Library binding
  - ○ Bindings for Python, Go, Rust, NodeJS, Bun, .NET.

- ★ **Project Background**
  - ○ Read about the birth of chDB auxten.com/the-birth-of-chdb
  - ○ Apache 2.0 Software License



```
import chdb
result  =  chdb.query(SQL)
```

Okay, in-process Database 🤔
What can I do with it?

Everything as a Table

SQLite PostgreSQL MySQL

Parquet

NumPy

CSV

chDB

DataFrame

JSON

Pyarrow

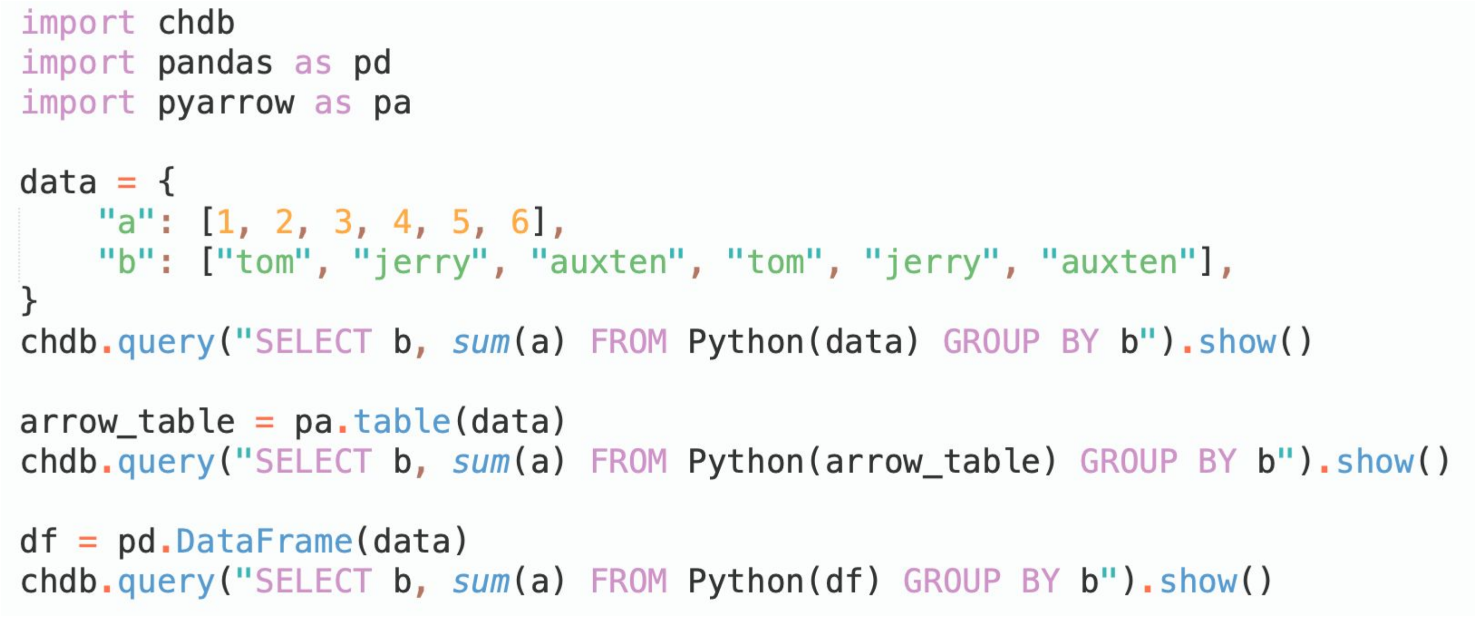
80+ formats

SQL Dump

PyReader

HTTP S3 HDFS

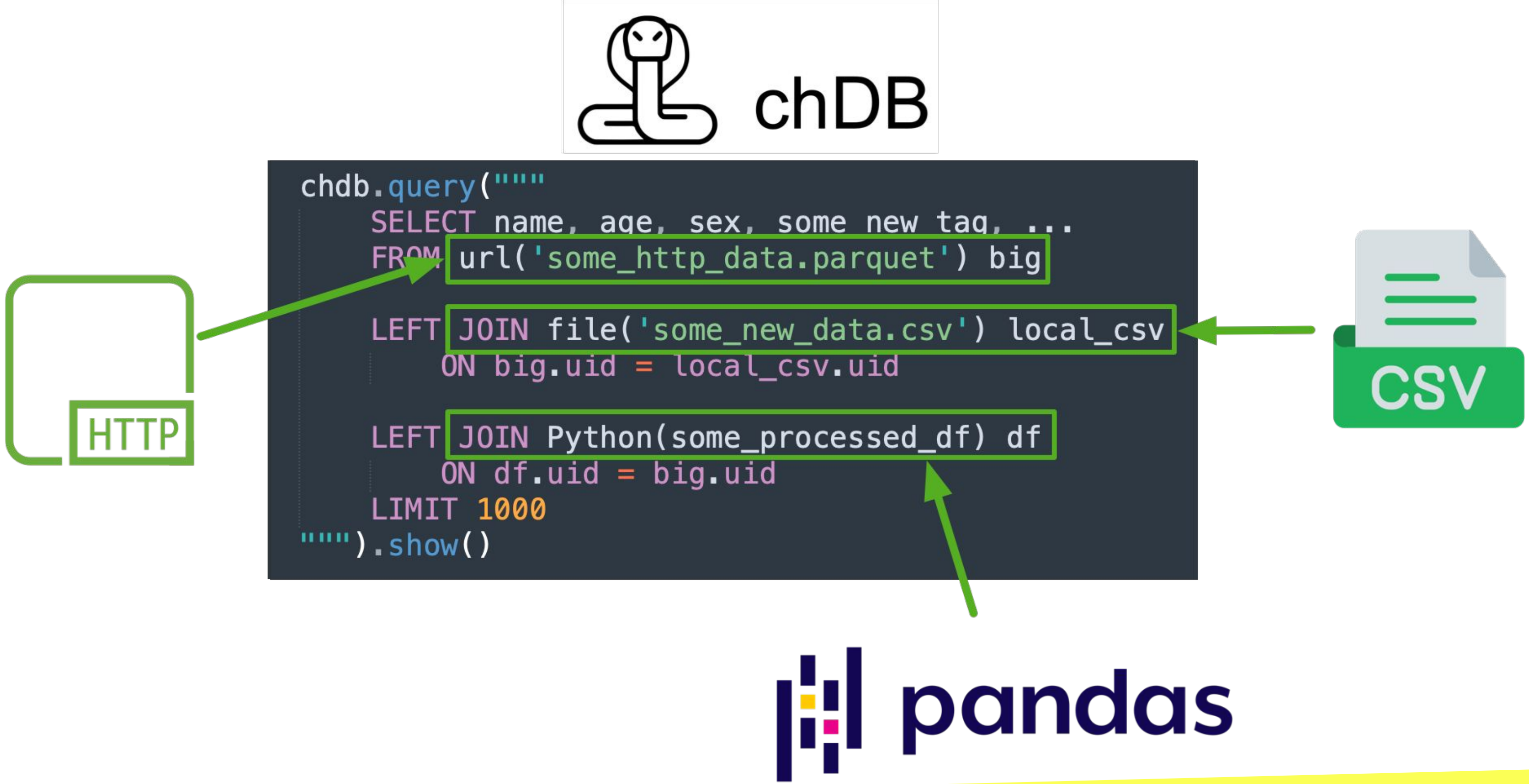
Query on Python Objects



```python
import chdb
import pandas as pd
import pyarrow as pa

data = {
    "a": [1, 2, 3, 4, 5, 6],
    "b": ["tom", "jerry", "auxten", "tom", "jerry", "auxten"],
}
chdb.query("SELECT b, sum(a) FROM Python(data) GROUP BY b").show()

arrow_table = pa.table(data)
chdb.query("SELECT b, sum(a) FROM Python(arrow_table) GROUP BY b").show()

df = pd.DataFrame(data)
chdb.query("SELECT b, sum(a) FROM Python(df) GROUP BY b").show()
```

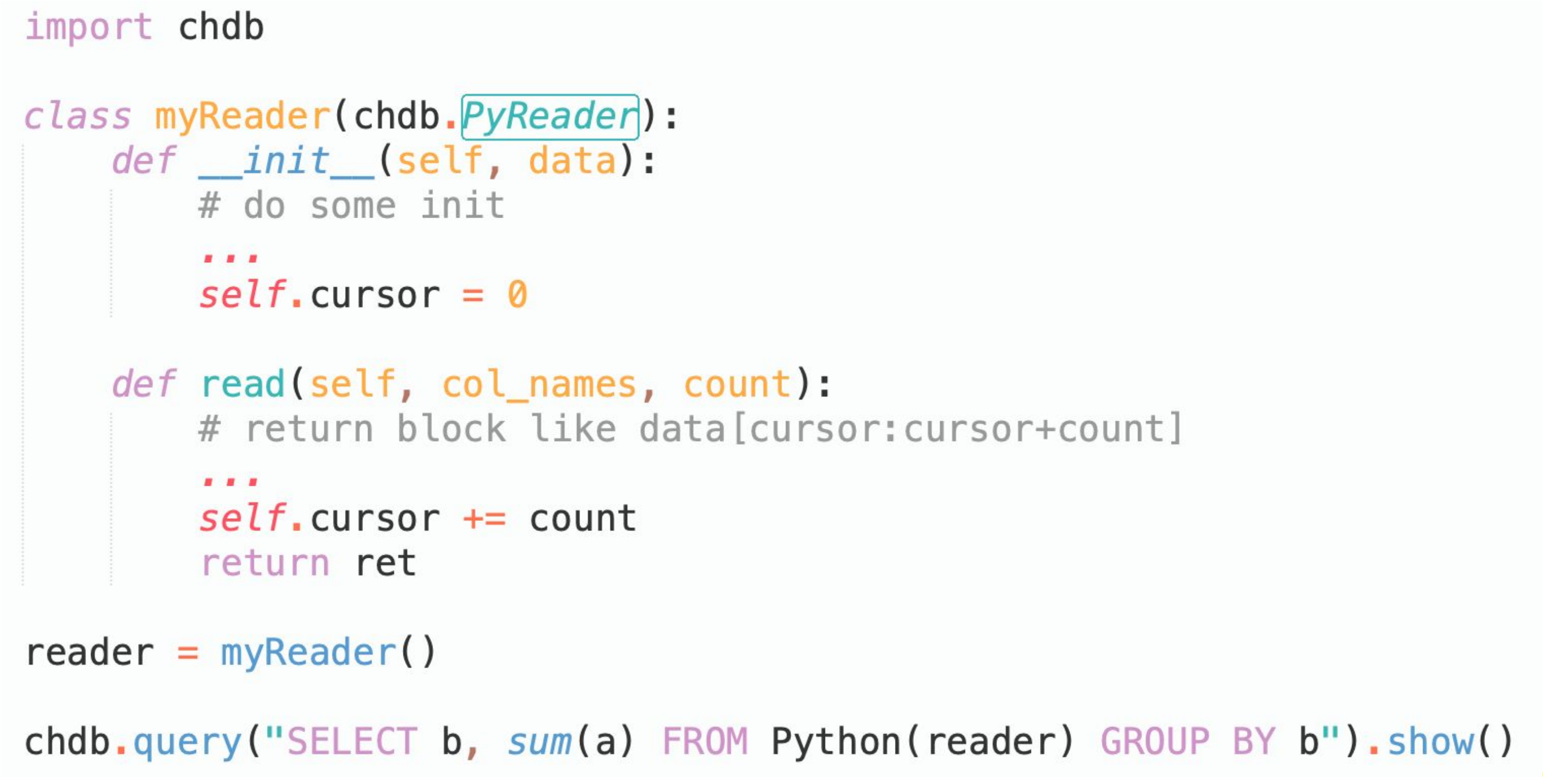
Only Numerical and String column type supported on v2.0.0b1

Join multiple data sources



```
chdb.query("""
    SELECT name, age, sex, some_new_tag, ...
    FROM url('some_http_data.parquet') big

    LEFT JOIN file('some_new_data.csv') local_csv
        ON big.uid = local_csv.uid

    LEFT JOIN Python(some_processed_df) df
        ON df.uid = big.uid
    LIMIT 1000
""").show()
```

Your own Table Engine in Python



```
import chdb

class myReader(chdb.PyReader):
    def __init__(self, data):
        # do some init
        ...
        self.cursor = 0

    def read(self, col_names, count):
        # return block like data[cursor:cursor+count]
        ...
        self.cursor += count
        return ret

reader = myReader()

chdb.query("SELECT b, sum(a) FROM Python(reader) GROUP BY b").show()
```

GIL

chDB

★ **chDB** in **Python**/Golang/Rust/NodeJS/Bun/.NET

python

```
import chdb
res = chdb.query('select version()', 'CSV');
```

★

_chdb.python-xxx.so

node JS

npm

```
const chdb = require("chdb-node");
var result = chdb.Execute("SELECT version()", "CSV");
```

Golang

```
package main

import (
    "github.com/chdb-io/chdb-go/chdb"
)

func main() {
    result := chdb.Query("SELECT version()", "CSV")
}
```

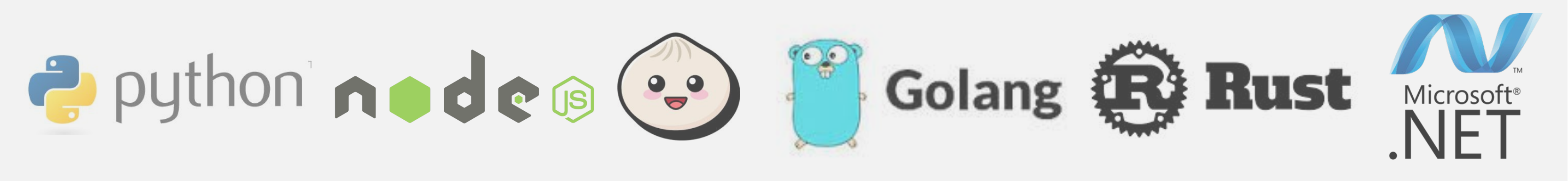
Microsoft .NET

Rust

libchdb.so

chDB

★ **Use chDB** (almost) **anywhere**



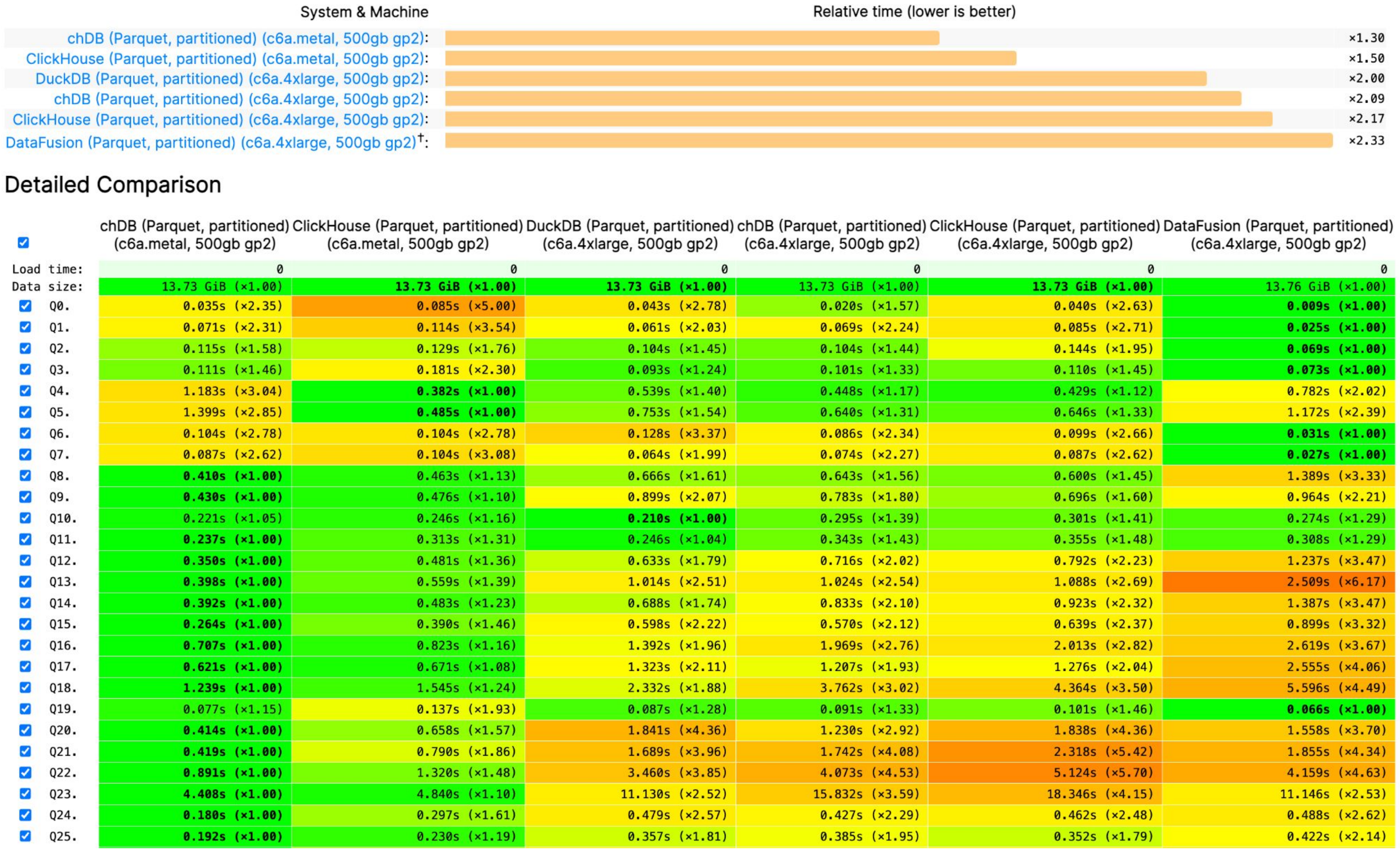
On

Data Science / LLM / Lambda / Mobile Phone

jupyter
AWS Lambda

Okay, Database 🤔
Is it fast?

Benchmark on Parquet



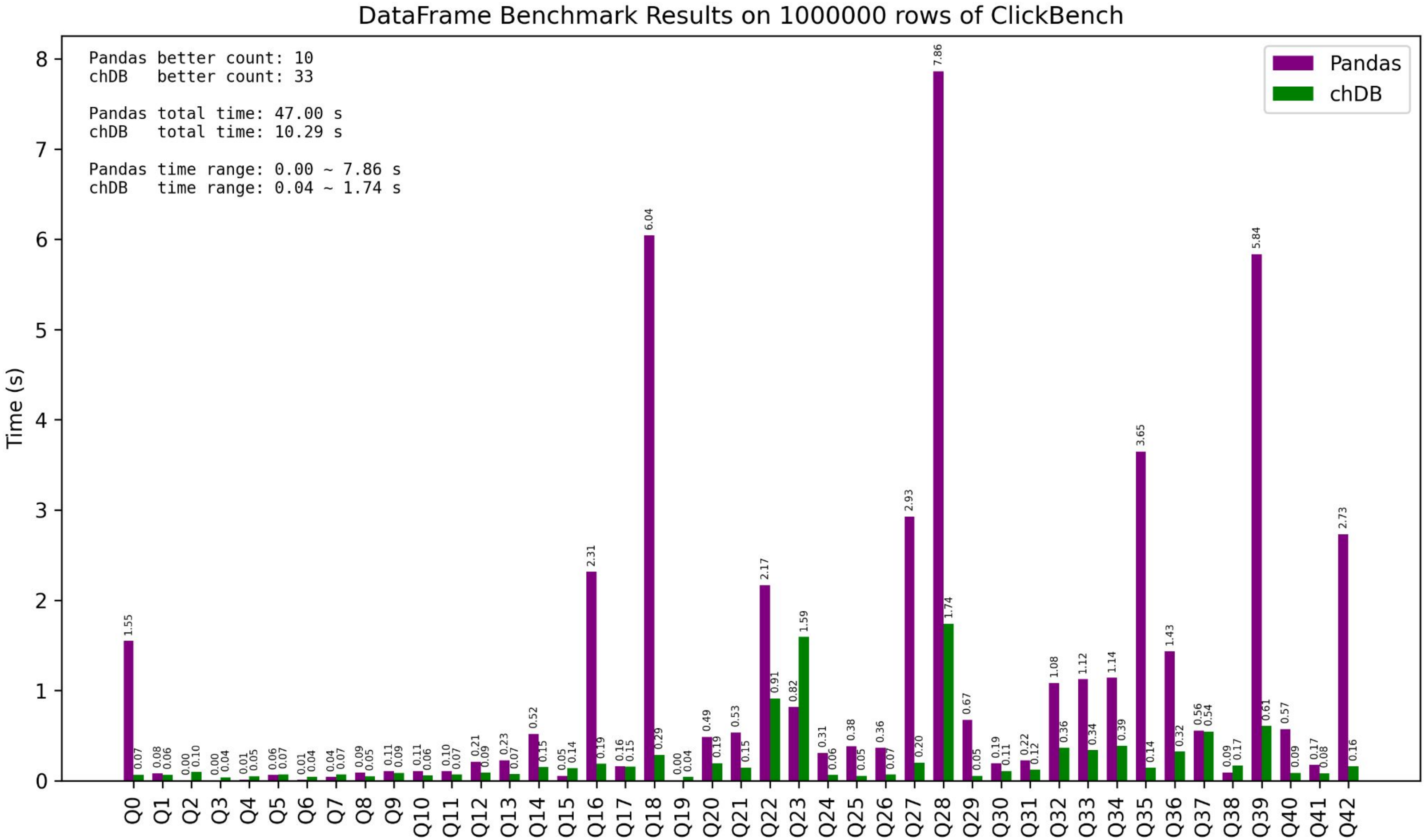
System & Machine	Relative time (lower is better)
chDB (Parquet, partitioned) (c6a.metal, 500gb gp2):	×1.30
ClickHouse (Parquet, partitioned) (c6a.metal, 500gb gp2):	×1.50
DuckDB (Parquet, partitioned) (c6a.4xlarge, 500gb gp2):	×2.00
chDB (Parquet, partitioned) (c6a.4xlarge, 500gb gp2):	×2.09
ClickHouse (Parquet, partitioned) (c6a.4xlarge, 500gb gp2):	×2.17
DataFusion (Parquet, partitioned) (c6a.4xlarge, 500gb gp2)†:	×2.33

Detailed Comparison

☑		chDB (Parquet, partitioned) (c6a.metal, 500gb gp2)	ClickHouse (Parquet, partitioned) (c6a.metal, 500gb gp2)	DuckDB (Parquet, partitioned) (c6a.4xlarge, 500gb gp2)	chDB (Parquet, partitioned) (c6a.4xlarge, 500gb gp2)	ClickHouse (Parquet, partitioned) (c6a.4xlarge, 500gb gp2)	DataFusion (Parquet, partitioned) (c6a.4xlarge, 500gb gp2)
Load time:		0	0	0	0	0	0
Data size:		13.73 GiB (×1.00)	**13.73 GiB (×1.00)**	**13.73 GiB (×1.00)**	13.73 GiB (×1.00)	**13.73 GiB (×1.00)**	13.76 GiB (×1.00)
☑	Q0.	0.035s (×2.35)	0.085s (×5.00)	0.043s (×2.78)	0.020s (×1.57)	0.040s (×2.63)	**0.009s (×1.00)**
☑	Q1.	0.071s (×2.31)	0.114s (×3.54)	0.061s (×2.03)	0.069s (×2.24)	0.085s (×2.71)	**0.025s (×1.00)**
☑	Q2.	0.115s (×1.58)	0.129s (×1.76)	0.104s (×1.45)	0.104s (×1.44)	0.144s (×1.95)	**0.069s (×1.00)**
☑	Q3.	0.111s (×1.46)	0.181s (×2.30)	0.093s (×1.24)	0.101s (×1.33)	0.110s (×1.45)	**0.073s (×1.00)**
☑	Q4.	1.183s (×3.04)	**0.382s (×1.00)**	0.539s (×1.40)	0.448s (×1.17)	0.429s (×1.12)	0.782s (×2.02)
☑	Q5.	1.399s (×2.85)	**0.485s (×1.00)**	0.753s (×1.54)	0.640s (×1.31)	0.646s (×1.33)	1.172s (×2.39)
☑	Q6.	0.104s (×2.78)	0.104s (×2.78)	0.128s (×3.37)	0.086s (×2.34)	0.099s (×2.66)	**0.031s (×1.00)**
☑	Q7.	0.087s (×2.62)	0.104s (×3.08)	0.064s (×1.99)	0.074s (×2.27)	0.087s (×2.62)	**0.027s (×1.00)**
☑	Q8.	**0.410s (×1.00)**	0.463s (×1.13)	0.666s (×1.61)	0.643s (×1.56)	0.600s (×1.45)	1.389s (×3.33)
☑	Q9.	**0.430s (×1.00)**	0.476s (×1.10)	0.899s (×2.07)	0.783s (×1.80)	0.696s (×1.60)	0.964s (×2.21)
☑	Q10.	0.221s (×1.05)	0.246s (×1.16)	**0.210s (×1.00)**	0.295s (×1.39)	0.301s (×1.41)	0.274s (×1.29)
☑	Q11.	**0.237s (×1.00)**	0.313s (×1.31)	0.246s (×1.04)	0.343s (×1.43)	0.355s (×1.48)	0.308s (×1.29)
☑	Q12.	**0.350s (×1.00)**	0.481s (×1.36)	0.633s (×1.79)	0.716s (×2.02)	0.792s (×2.23)	1.237s (×3.47)
☑	Q13.	**0.398s (×1.00)**	0.559s (×1.39)	1.014s (×2.51)	1.024s (×2.54)	1.088s (×2.69)	2.509s (×6.17)
☑	Q14.	**0.392s (×1.00)**	0.483s (×1.23)	0.688s (×1.74)	0.833s (×2.10)	0.923s (×2.32)	1.387s (×3.47)
☑	Q15.	**0.264s (×1.00)**	0.390s (×1.46)	0.598s (×2.22)	0.570s (×2.12)	0.639s (×2.37)	0.899s (×3.32)
☑	Q16.	**0.707s (×1.00)**	0.823s (×1.16)	1.392s (×1.96)	1.969s (×2.76)	2.013s (×2.82)	2.619s (×3.67)
☑	Q17.	**0.621s (×1.00)**	0.671s (×1.08)	1.323s (×2.11)	1.207s (×1.93)	1.276s (×2.04)	2.555s (×4.06)
☑	Q18.	**1.239s (×1.00)**	1.545s (×1.24)	2.332s (×1.88)	3.762s (×3.02)	4.364s (×3.50)	5.596s (×4.49)
☑	Q19.	0.077s (×1.15)	0.137s (×1.93)	0.087s (×1.28)	0.091s (×1.33)	0.101s (×1.46)	**0.066s (×1.00)**
☑	Q20.	**0.414s (×1.00)**	0.658s (×1.57)	1.841s (×4.36)	1.230s (×2.92)	1.838s (×4.36)	1.558s (×3.70)
☑	Q21.	**0.419s (×1.00)**	0.790s (×1.86)	1.689s (×3.96)	1.742s (×4.08)	2.318s (×5.42)	1.855s (×4.34)
☑	Q22.	**0.891s (×1.00)**	1.320s (×1.48)	3.460s (×3.85)	4.073s (×4.53)	5.124s (×5.70)	4.159s (×4.63)
☑	Q23.	**4.408s (×1.00)**	4.840s (×1.10)	11.130s (×2.52)	15.832s (×3.59)	18.346s (×4.15)	11.146s (×2.53)
☑	Q24.	**0.180s (×1.00)**	0.297s (×1.61)	0.479s (×2.57)	0.427s (×2.29)	0.462s (×2.48)	0.488s (×2.62)
☑	Q25.	**0.192s (×1.00)**	0.230s (×1.19)	0.357s (×1.81)	0.385s (×1.95)	0.352s (×1.79)	0.422s (×2.14)

t.co/nb75mvEyyO

Benchmark on DataFrame – 4.6x Faster



t.co/uKjIIqL04M

Okay, But WHY?

Why chDB is Fast

Mostly, ClickHouse is Fast

Why chDB is Fast

Just make sure

Python

does not slow it down

1. Hold this → GIL
2. Do everything with C++ in Parallel

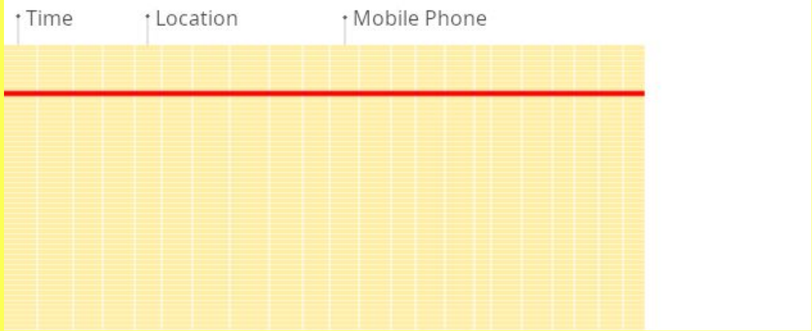
Okay, But why is ClickHouse Fast?

Why ClickHouse is Fast

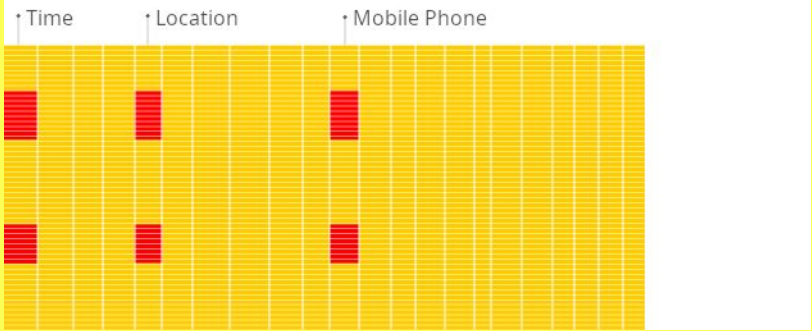
- Column-oriented storage
- Data compression
- Vectorized query execution
- JIT(Just In Time) & Dynamic Dispatch
  - Compile SQL into native cpu instruction
  - Runtime check CPU spec and dispatch to AVX, AVX2, AVX512 specialized function
- ......
- Keep benchmark and optimization for 15 years

https://clickhouse.com/docs/en/faq/general/why-clickhouse-is-so-fast

Row-oriented databases



Column-oriented databases



Recap

chDB

Features & Use-Cases

🚀 Pure Performance

In-process chDB eliminates overhead communication between clients and servers accessing cloud datasets

🧩 Seamless Integration

Full ClickHouse OLAP functionality included, no need to change query style or renounce any advanced feature

💡 Reduced Consumption

chDB runs alongside your code on-demand, with no need to maintain any costly backend server infrastructure

📈 Real-time Analytics

In-process chDB enables OLAP analytics directly off cloud storage, S3, Parquet files or ClickHouse services

🧪 Quick Prototype

Develop and showcase your prototype directly in your Notebook on Laptop

chDB

Just try chDB

Docs

- For chDB specific examples and documentation refer to clickhouse.com/docs/en/chdb
- For SQL syntax, please refer to ClickHouse SQL Reference
- The birth of chDB auxten.com/the-birth-of-chdb

Demos

- Project Documentation and Usage Examples
- Colab Notebooks and other Script Examples
- ClickBench of embedded engines

Contact

- Email: auxten@clickhouse.com
- Twitter: @auxten

bitly

Thank You!

Check it out → chdb.io