< >

COM 617 User Guide

Design Document (Template issue: 1.0)

Enter Author Name: Luke Wood, Iona Pitt, Josh Clarke, Kyle Roberts, Danny Agha Report Date: 25/04/24

List of contents

[**1 Setup [1]** 4](#_Toc10454)

[1.1 Installing and Using this Repo 4](#_Toc10455)

[1.2 Configuring ClickHouseDB 5](#_Toc10456)

[1.3 Configuring PostgreSQL 6](#_Toc10457)

[1.4 Configuring TimescaleDB 7](#_Toc10458)

[1.5 Configuring ArcticDB 8](#_Toc10459)

[1.6 Troubleshooting 8](#_Toc10460)

[**2** Page Interface 9](#_Toc10461)

[2.1 Home Page 9](#_Toc10462)

[2.2 Database Pages 9](#_Toc10463)

# **Setup [1]**

## Installing and Using this Repo

This project has four main components initially,

1. the Flask/Plotly/Dash python module.
2. the Postgres Docker container
3. the two Clickhouse DB containers

The steps to install and run it are:

Git clone this repo then create the virtual environment and install the packages:

*cd db\_bench python -m venv .venv pip install -r requirements.txt*

Create local versions of the config files

*cp .vscode/launch.json.example .vscode/launch.json cp .env\_example .env*

*cp postgres.env\_example postgres.env*

Enable all user permissions

*cd etc\clickhouse-server*

In *chuser.xml,* add the grant to the user profile:

*<chuser>*

*<profile>ch\_profile</profile>*

*<networks>*

*<ip>::/0</ip>*

*</networks>*

*<password>chuser\_pwd</password>*

*<quota>ch\_quota</quota>*

*<grants>*

*<query>GRANT ALL ON \*.\*</query>*

*</grants>*

*</chuser>*

This will create a *subdirectory .venv* containing a virtual Python environment isolating the project from other projects on your computer. You may want to move across to using the poetry package manager as one of your deliverables. It handles dependencies in a more intelligent way than venv and pip.

If you're using VS Code, note the .*vscode* directory which contains an entry allowing you to start and debug the project.

## Configuring ClickHouseDB

You can try this now but will likely get errors about not being able to connect to the database. So the next step is to run up the Docker containers for Clickhouse and configure them. You will need Docker Desktop installed on your machine.

(The following commands are to be entered in your machines terminal) *cd db\_bench*

*docker-compose up ch\_server ch\_client*

This will build your containers and run them locally. You can see their status with docker container ls -a.

Now we need to check that the clickhouse database is running locally, choose your preferred SQL client. I like to use DBeaver. Create a connection of type Clickhouse on localhost, port 8124 (specified in *docker-compose.yml*), user *chuser* and password *chuser\_pwd* (specified in */etc/clickhouse-server/users.d/chuser.xml* and *.env*) and we start with database default.

You should now be able to connect to your locally running Clickhouse docker container. When you are connected, open an SQL terminal and create the database. Disconnect and reconnect as this will refresh DBeaver - the new database will not show up on the GUI if you don't do this.

Create a new SQL script and run the below command;

*CREATE DATABASE ts\_db;*

Now create the demo timeseries table with the following SQL command. This only creates a small table. Once you're sure of the installation, change all the t*oDate(2021* to *toDate(2022* to generate a year and 10 minute's worth of 1 second time series data. Once again, refresh DBeaver.

*CREATE TABLE ts\_db.demo\_ts*

*ENGINE = MergeTree*

*ORDER BY tuple()*

*AS*

*SELECT toDateTime(arrayJoin(range(toUInt32(toDateTime('2021-01-01 00:00:00')), toUInt32(toDateTime('2022-01-01 00:10:00')), 1) )) as cdatetime,*

*toSecond(toDateTime(arrayJoin(range(toUInt32(toDateTime('2021-01-01 00:00:00')), toUInt32(toDateTime('2022-01-01 00:10:00')), 1) ))) +*

*toMinute(toDateTime(arrayJoin(range(toUInt32(toDateTime('2021-01-01 00:00:00')), toUInt32(toDateTime('2022-01-01 00:10:00')), 1) ))) +*

*2 \* toHour(toDateTime(arrayJoin(range(toUInt32(toDateTime('2021-01-01 00:00:00')), toUInt32(toDateTime('2022-01-01 00:10:00')), 1) ))) +*

*5 \* toDayOfWeek(toDateTime(arrayJoin(range(toUInt32(toDateTime('2021-01-01*

*00:00:00')), toUInt32(toDateTime('2022-01-01 00:10:00')), 1) ))) +*

*8 \* toWeek(toDateTime(arrayJoin(range(toUInt32(toDateTime('2021-01-01 00:00:00')), toUInt32(toDateTime('2022-01-01 00:10:00')), 1) ))) +*

*12 \* toMonth(toDateTime(arrayJoin(range(toUInt32(toDateTime('2021-01-01*

*00:00:00')), toUInt32(toDateTime('2022-01-01 00:10:00')), 1) ))) +*

*20 \* (toYear(toDateTime(arrayJoin(range(toUInt32(toDateTime('2021-01-01*

*00:00:00')), toUInt32(toDateTime('2022-01-01 00:10:00')), 1) )))-2021) as ts\_values*

Make sure all the packages in chdemoapp.py have been installed, and then you can start the app and it should connect to the ClickHouse database and show some data. This can now also be done with the db\_bench.py application.

## Configuring PostgreSQL

To configure Postgres, run the command docker compose up db. This will create the *psql\_db* container. Go to DBeaver and create a new connection to a Postgres database on port 5432 with the username *postgres* and password *postgres*.

Once connected, create a table with the SQL command

*CREATE TABLE demo\_ts ( cdatetime DATE, ts\_values INTEGER*

);

and generate some data with

*WITH time\_series AS (*

*SELECT \* FROM generate\_series(*

*'2021-01-01 00:00:00'::timestamp,*

*'2022-01-01 00:10:00'::timestamp,*

*'1 second'::interval ) as cdatetime*

),

*random\_values AS (*

*SELECT random() \* 100 AS ts\_values* -- Adjust range as needed

*FROM generate\_series(1, 5)* -- Generate 5 random values

*)*

*INSERT INTO demo\_ts (cdatetime, ts\_values)*

*SELECT time\_series.cdatetime, random\_values.ts\_values*

*FROM time\_series*

*CROSS JOIN random\_values;*

Lastly, in order to display the data on the Streamlit app, navigate to your *.streamlit* folder (default is at *C:\Users\Username\.streamlit)* and create a *secrets.toml* file. Add the following code:

*CREATE TABLE demo\_ts (* *cdatetime DATE, ts\_values INTEGER*

*);*

And generate some data with

*WITH time\_series AS (*

*SELECT \* FROM generate\_series(*

*'2021-01-01 00:00:00'::timestamp,*

*'2022-01-01 00:10:00'::timestamp,*

*'1 second'::interval*

*) as cdatetime*

*),*

*random\_values AS (*

*SELECT random() \* 100 AS ts\_values -- Adjust range as needed*

*FROM generate\_series(1, 5) -- Generate 5 random values*

*)*

*INSERT INTO demo\_ts (cdatetime, ts\_values)*

*SELECT time\_series.cdatetime, random\_values.ts\_values*

*FROM time\_series*

*CROSS JOIN random\_values;*

## Configuring TimescaleDB

To configure Timescale, run the command docker compose up timescaledb. This will create the *tmscl\_db* container. Go to DBeaver and create a new connection to a Timescale database on *port 5433* with the username *postgres* and password *postgres*. (Timescale uses Postgres) Once connected, create a table with the SQL command.

*CREATE TABLE demo\_ts ( cdatetime DATE, ts\_values INTEGER*

*);*

and generate some data with

*WITH time\_series AS (*

*SELECT \* FROM generate\_series(*

*'2021-01-01 00:00:00'::timestamp,*

*'2021-06-01 00:10:00'::timestamp,*

*'1 second'::interval ) as cdatetime*

*),*

*random\_values AS (*

*SELECT random() \* 100 AS ts\_values* -- Adjust range as needed

*FROM generate\_series(1, 5)* -- Generate 5 random values

*)*

*INSERT INTO demo\_ts (cdatetime, ts\_values)*

*SELECT time\_series.cdatetime, random\_values.ts\_values*

*FROM time\_series*

*CROSS JOIN random\_values;*

## Configuring ArcticDB

Make sure Clickhouse DB is set up before configuring the ArcticDB database To first install ArcticDB locally, run the command *pip install arcticdb*.

Create an Amazon AWS Account and set up an S3 bucket. Within the project *.env* file, add the URL for the S3 bucket

'*s3s://s3.<REGION\_NAME>.amazonaws.com:<BUCKET\_NAME>?aws\_auth=tru e*' to ARCTIC\_URL.

To access the S3 bucket, IAM User Access can be set up on AWS. See https://docs.arcticdb.io/4.4.1/#getting-started for more information. The other option which is less recommended is making the bucket publically available by disabling the 'Block public access' settings and adding a statement to the 'Bucket Display'. These options can be found within the AWS bucket options.

Public Bucket Policy:

*{* *"Id": "BucketPolicy", "Version": "2012-10-17", "Statement": [ { "Sid": "AllAccess",*

*"Action": "s3:", "Effect": "Allow", "Resource": [*

*"arn:aws:s3:::<BUCKET\_NAME>", "arn:aws:s3:::<BUCKET\_NAME>/" ],*

*"Principal": "\*"* *} ] }*

Run the *arcticdb\_setup.py* file by running *python .\arcticdb\_setup.py* from the root folder (This may take some time). This sends the same dataset from the Clickhouse database to the Arctic storage.

## Troubleshooting

ImportError: cannot import name *'load\_dotenv'* from '*dotenv'*

If you get the error message shown above, install the package *python\_dotenv* instead of *dotenv.* You do not need to change the import name, as dotenv will automatically be installed with *python\_dotenv.*

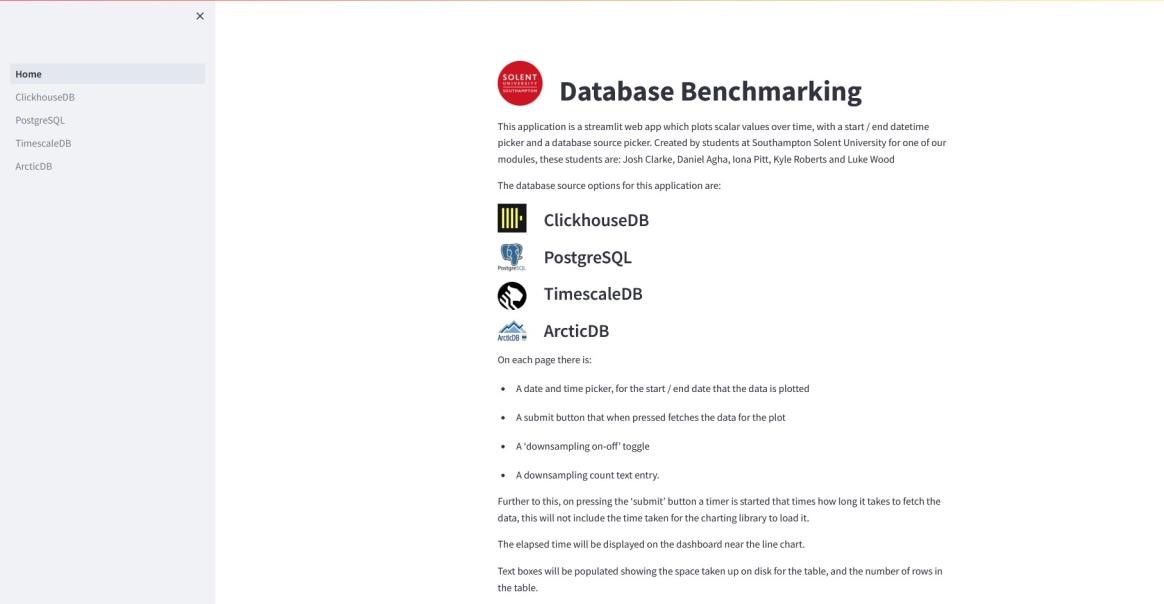
*toml.decoder.TomlDecodeError:* Key group not on a line by itself. (Line 1 column 1 char 0)

If you get the error message shown above, go to your *.streamlit* folder on your computer (*default is at C:\Users\Username\.streamlit*) and delete the *config.toml* file.

# Page Interface

## Home Page

The home page has a brief introduction and run down of the project as well as navigation buttons for the 4 other pages that allow you access each database benchmarking page.



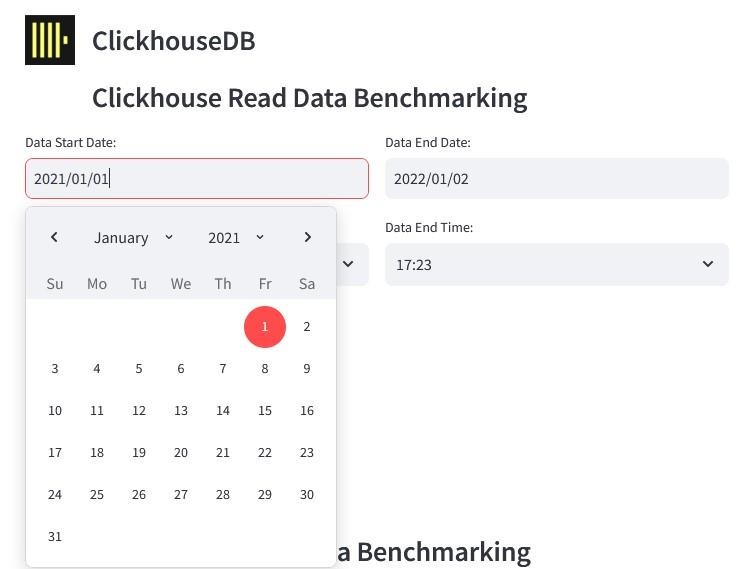
Navigation Panel



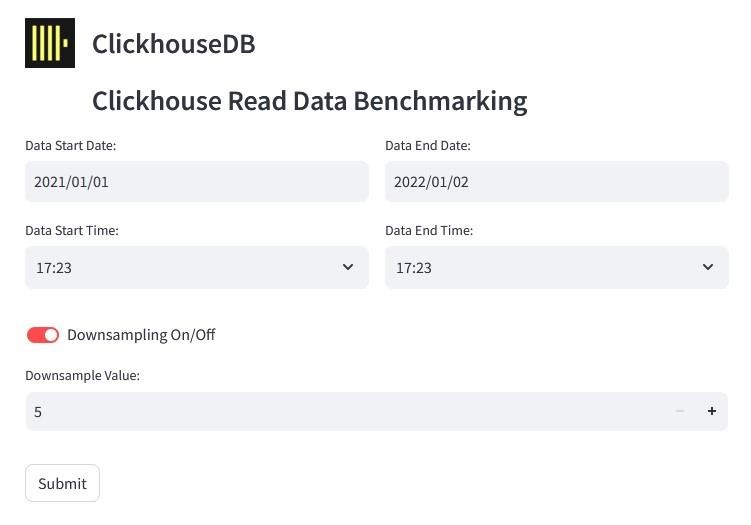
## Database Pages

Each database page will consist of:

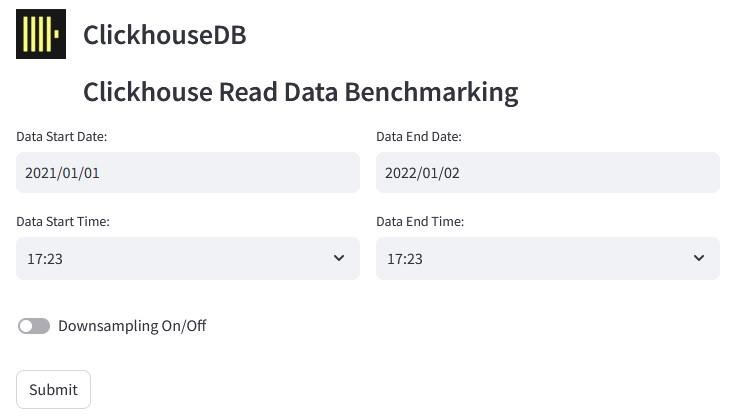
* Date and time pickers (start and end of the database sample)



* The downsampling on/off toggle (this will show the downsampling count text entry) Off:



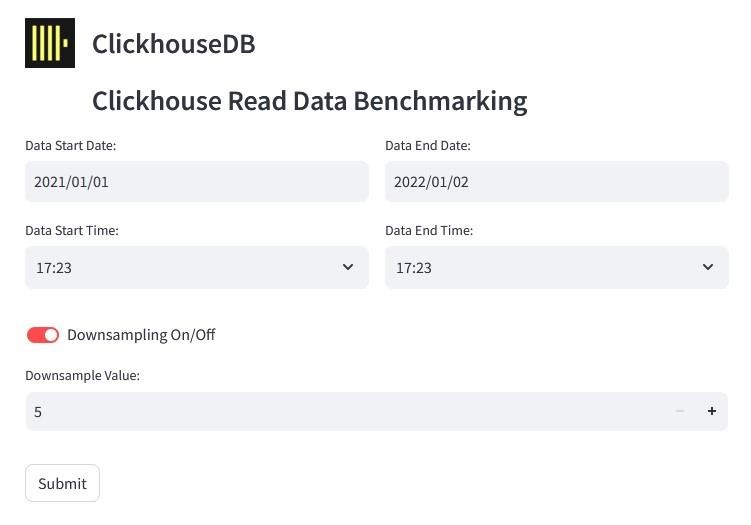
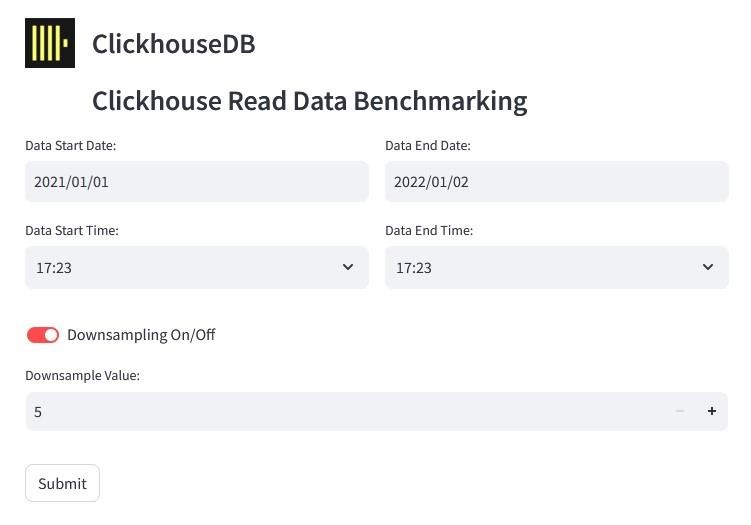
On:



* The downsampling count text entry. (Limited to 5000 samples)

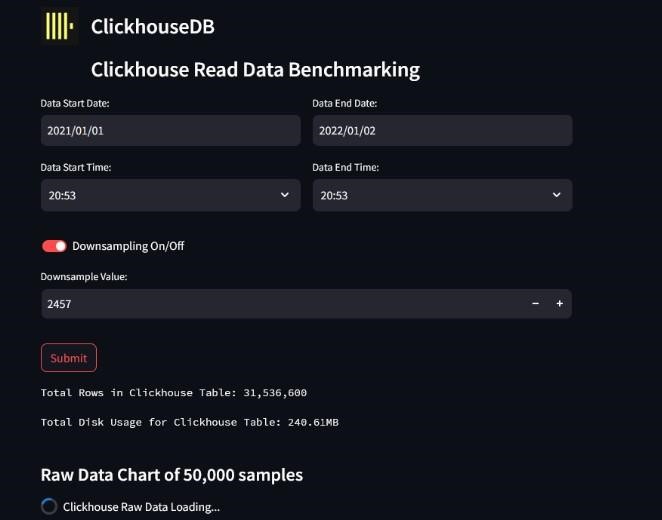
-

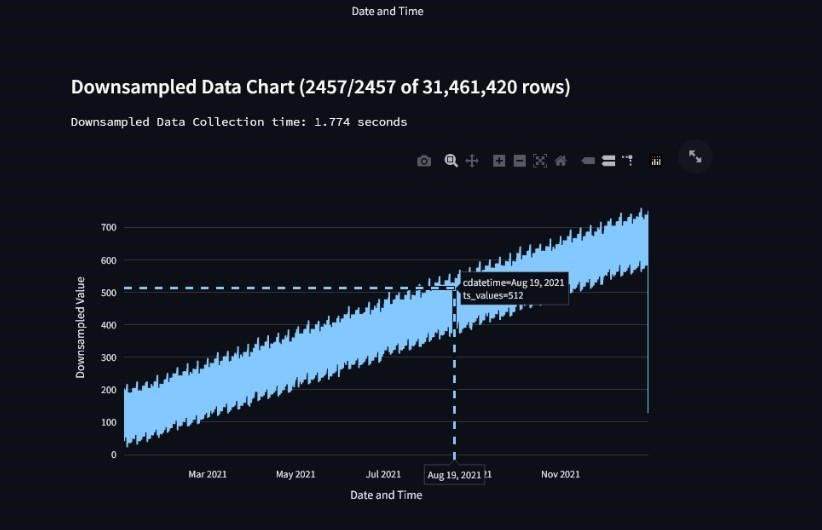
The submit option that when pressed, will begin the process of fetching the data



Once the submit button has been pressed:

* A timer starts, that times how long it takes to fetch the data (this does not include the time taken for the charting library to display the data on the graph).
* The elapsed time which is displayed on the dashboard near the line chart.
* Text boxes populated, showing the space taken up on disk for the table and the number of rows in the table.
* A text box showing the total database size on disk in MB..
* A graph of the fetched data.





Page 14