Group 3 – Database Benchmarking

Group Members: Danial Agha, Iona Pitt, Joshua Clarke, Kyle Roberts, Luke Wood Supporting Tutor: Martin Reed Project Sponsor: Nick Thorne GitHub: https://github.com/NickThorne123/db\_bench

COM617 – Industrial Consulting Project

2024

Contents

[Introduction 3](#_Toc165322407)

[Project Objectives 3](#_Toc165322408)

[Requirements 3](#_Toc165322409)

[Scope and Exclusions 5](#_Toc165322410)

[Monitoring and Evaluation 6](#_Toc165322411)

[Project Organisation and Structure 6](#_Toc165322412)

[Project Milestones and Management 7](#_Toc165322413)

[Implementation 7](#_Toc165322414)

[Results 8](#_Toc165322415)

[Conclusion 8](#_Toc165322416)

[References 8](#_Toc165322417)

[Appendices 8](#_Toc165322418)

[Appendix A – Design Document 8](#_Toc165322419)

[Abstract / Executive Summary 8](#_Toc165322420)

[List of contents 8](#_Toc165322421)

[Introduction 10](#_Toc165322422)

[Scope of the document 10](#_Toc165322423)

[Definitions, acronyms and abbreviations 10](#_Toc165322424)

[Conventions 11](#_Toc165322425)

[Stakeholders 12](#_Toc165322426)

[Architectural Design 14](#_Toc165322427)

[Detailed Design 17](#_Toc165322428)

[Design Document References / Bibliography 0](#_Toc165322429)

[Appendix B – User Guide 0](#_Toc165322430)

[List of contents 0](#_Toc165322431)

[Setup [1] 1](#_Toc165322432)

[Installing and Using this Repo 1](#_Toc165322433)

[Configuring ClickHouseDB 2](#_Toc165322434)

[Configuring PostgreSQL 3](#_Toc165322435)

[Configuring TimescaleDB 4](#_Toc165322436)

[Configuring ArcticDB 5](#_Toc165322437)

[Troubleshooting 5](#_Toc165322438)

[Page Interface 6](#_Toc165322439)

[Appendix C – ClickhouseDB Stats 10](#_Toc165322440)

[Appendix D – PostgreSQL Stats 10](#_Toc165322441)

[Appendix E – TimeseriesDB Stats 10](#_Toc165322442)

[Appendix F – ArcticDB Stats 10](#_Toc165322443)

[Appendix G – Sprint Overview 10](#_Toc165322444)

[Appendix H – Test Plan Document 11](#_Toc165322445)

# Introduction

The purpose of this COM617 project, is to compare different timeseries database technologies and benchmark them against each other, for the customer, a company called Rockstone Data Ltd. This is because, they have recently adopted a new column orientated database (C-Store) called ClickhouseDB and they would like a demonstrator on the companies’ website, showing how a C-Store compares to a ‘traditional’ database. These being PostgreSQL, TimescaleDB and ArcticDB with the aim of showing ClickhouseDB as a more effective database solution to previous, more ‘traditional’ databases.

# Project Objectives

The key objective of this project was to create a one page ‘dashboard’ web application, which plots time series graphs, chosen by the user using a start and end datetime picker, on database the user has chosen to benchmark the results. Upon a user pressing a submit button, this faces the data for the plot and displays the time taken to fetch the data, the disk space taken up retrieving the data, the number of rows in the table and the GB of disk storage per million rules. Finally, there will also be a down-sampling toggle, allowing a user to input a down-sampling count between 1-5000. The definition for down-sampling is in Appendix A.

# Requirements

As you can see from appendix I, a requirements table is provided and includes a complete list of the main requirements that where needed to mark the project as complete in the eyes of the team and the customer. Detailing things such as UI, database, docker, documentation and testing requirements.

# Scope and Exclusions

The main work streams that aided development of the project, are in-built features on our GitHub repository. The work streams, where issues, set on the projects Kanban board in the actions section of our repository.

The kanban board helped establish order with our tasks, enabling better collaboration and maximising our efficiency. This meant allowing the team, to visualise the tasks that needed to be completed, without completing unnecessary work.

These columns where: “Backlog” (items that have been started), “Ready” (ready to be picked up), “In-Progress” (actively being worked on), “In-Review” (completed and being checked by a team member) and finally “Done’ (item has been completed).

The tasks on the kanban board where dynamically reviewed every week and refined towards the end of each sprint, to keep the scope of the project under control and on track to meet the project requirements. This also allowed the team to remove and add tasks, meaning we can better complete the objectives of the sprint and increase customer satisfaction by the end-result. This is because the customer had the ability to view the board and see our progress in delivering their product.

Finally, to enable collaboration within the team, we opted to use Microsoft (MS) Teams, which offered the team file storage, video conferencing and a work messaging chat, meaning the team could keep in touch on a regular basis and get the tasks completed, within the scope of each sprint.

# Project Organisation and Structure

The work was split amongst the 5 group members, based on our strengths, in-order to complete the project as effectively as possible assigning each member job roles. Because the team was quite small it meant one person picked up multiple job roles.

First, the Customer, as mentioned above, was the company Rockstone Data Ltd, which provided the set of requirements to the team for what they envisage for the product.

The project manager and scrum master were combined, undertaken by Josh Clarke. They are also the teams full stack developer as they have the most experience with Python and the Database side of the project.

Iona Pitt assumed the mantle of product owner and backend developer, because they have the most experience SCRUM methodologies, whilst also having some experience in working with databases.

Danial Agha’s responsibility was the software tester, this involved completing the UI testing and getting the CI/CD pipeline up and running for the repository. The reason for this is because they have previous experience on projects testing the software.

Kyle Roberts took on the role of UI developer. Again, this is because their strengths lie in creating UIs as it is one of their main job roles. They are also involved in generating documentation for the project.

Finally, Luke Wood oversees customer relations. Meaning they, are the point of contact for the customer and most of the questions from the team to the customer go through them. Not only this but they also worked on documentation too.

# Project Milestones and Management

The project was broken down into three sprints, detailed in Appendix H. These lasted 3 weeks, 4 weeks and 8 weeks.

To manage the project, the project manager Josh implemented two weekly meetings. The first one, on Monday’s at 5pm, was a quick 15 minute catch up, to discuss what each team member has been doing and what they will be doing during the week. The next meeting was Thursdays at 3pm, this lasted from anywhere between 30 minutes to 2 hours and was the main weekly meeting. This allowed us to help each other if needed, a catch up with our support tutor, another catch up meeting to dynamically review the work within the scope of the sprint.

As shown in appendix h, the project milestones were written for us, giving the project structure, providing goals for the end of each sprint. The first sprint, was 3 weeks long and the main goal to the gather the projects requirements, detailed in the requirements section and was completed by 9th February.

The next sprint was the initial proof of concept (PoC), as agreed with the customer, this meant a very basic application with the required functionality, on only one database. Because this was a PoC, it meant the way it looked wasn’t as refined as the minimal viable product (MVP). This ended with a demonstration to the support tutor and customer, where they signed off that they were happy.

The final sprint was creating the MVP, this involved a functional application that provides enough features for the customer to build upon and implement themselves. This meant supplying them with the accompanying documentation.

# Monitoring and Evaluation

Evaluation was given by the customer at the end of sprint two, this is because, as mentioned in the project milestones and management section, our PoC was demonstrated to the customer, where they gave their feedback on what they liked about the product so far and how we can improve it.

Not only this, but because we could easily contact the customer, this allowed us to also gather constant feedback from them about different requirements and how it relates to the work that needed completing. This really helped the team as they gave us feedback on multiple issues such as, feedback on our UI design, issue with setting up our work environments and issues to overcome with each database.

On top of this, when new code/ functionality was created for the programme, it had to be checked by another developer to make sure that it operated correctly, which in turn provided a fresh set of eyes to catch any errors that could have been missed by the developer creating the new code. This meant the new code was both monitored and evaluated before being merged into our master branch.

The final way the project was monitored, is through appendix H, the test plan document. As detailed in appendix H, these are automatic unit tests that make up part of the CI/CD pipeline and prevent new code being merged into the repository. This shows to the customer the tests that have either passed or failed and demonstrates to the customer that the team has produced a high-quality product for them, by identify any bugs.

# Implementation

At the start of the project, a standards document was created to give the team guidance on how to name their branch, with coding and documentation conventions. This was to increase speed and efficiency in development by improving code readability and how it is maintained. Also, the ‘Joel Test’ was applied, to the quality of your software team by providing 12 yes-or-no questions to determine whether your programming team is up to par (Amaresan, 2021). Although some of the principles didn’t apply, such as, do new candidates write code during their interviews.

The implementation of the product changed quite heavily from the initial designs to the MVP. This was due to conversations with the customer on features to remove/ keep and how best to make the UI more user friendly.

As shown in appendix a, compared to appendix b, the design of the original UI changed drastically. With the most notable change being the alteration from a slider to get the down-sampling number, to a text input (with plus and minus buttons) becoming more user friendly, as it would have been hard to get an exact number of values to reduce it by. Not only this but the removal of unnecessary buttons made the UI feel less cluttered.

One thing that didn’t change much from appendix a to the MVP, was the class structure on how to load each data into the application. It followed a simple structure of creating a class for each database we were planning on using, using a function to fetch the data and inputting that into their respective graphs and displaying of each calculation.

Finally, some databases only allowed a user to create a certain amount of data points before running into issue such as taking up too much disk space. A work around for this, which was agreed with the customer, was to take a three-month sample size (as opposed to a year) and multiple each benchmarking calculation by four to show it as if it was a years’ worth.

# Results

To make sure we were producing results for the customer, as previously eluded too, we had regular contact with the customer providing opportunities and feedback on what they want to get out of the product we will deliver to them. Not only this, but as part of the set up for the project, the customer was kind enough to lend us a repository for code allowing use to enhance our workflows and provide version control capabilities.

Because the repository was owned by the customer, it meant that, they had direct oversight over pull requests, the kanban board and documentation, allowing to raise issues faster and checkout the codes functionality for themselves.

Another way results where shown was through the requirements table detailed in appendix I. As you can see, there are two main columns, the ‘Requirement’ column, this provides a description of what needs to be implemented to show the requirement as complete and the ‘Completed?’ column. This has a simple yes/ no structure to show if the requirement has been met. There is also a non-applicable column, to show if the original requirement has been removed or the customer has given permission to not complete that requirement.

Finally, results can be measured with the stats produced by the benchmarking data, seen in appendix C through to F. As shown, from the stats, ClickhouseDB comes out on top in almost all metrics when compared to the, more ‘traditional’ databases, except RAM usage when trying to read the data, with it coming in third behind TimescaleDB and PostgreSQL. Given the data for each database was over the same amount of time, it shows how effective a c-store is compared to a traditional database, taking up less disk usage and the least time to read and write data, to and from the database.

# Conclusion

// Summary and presentation of the final product/artefact/solution and recommendations for any further work

In conclusion, the team believes this project to have been a success in not only proving, for the customer, that ClickhouseDB is a more effective database solution compared to ‘traditional’ databases but also the quality of work that was produced by the team. This is because, the team stuck to the project plan and the SCRUM methodology, keeping communication clear between not only the team but also the customer. We also met almost all the requirement set out by the customer with exceptions to a few of the stretch goals because, we ran out of time to implement them.

However, I think a big place we could improve on in the future is when gathering the requirements, keeping them as close as possible to the original requirements set out and not allowing the customer to dictate requirements mid-development. Not only this, but due to other workloads, it meant we missed events the customer invited us too. As a team we feel these could have been beneficial to allowing closer ties with the customer and that more effort could have been made to be free during these events.

Finally, as team, this was a project we enjoyed working on and felt the team ‘gelled’ well. This allowed every team member to have a role that suited their strengths but also allowed each member an opportunity to work on their weaknesses and improve overall, as software engineers.

WC – 2143(30/04/24 @ 0015)

2200 – 2143 = max words left: 57

# References

AMARESAN, S., 2021. The 2021 Guide to The Joel Test for Programming In: Swetha Amaresan's weblog. 04 February 2021 [viewed 29 April 2024]. Available from: <https://blog.hubspot.com/service/joel-test>

# Appendices

## Appendix A – Design Document

### Abstract / Executive Summary

Rockstone Data is a software application development company founded in 2018 by Nick Thorne.

The company has recently utilised a new column-orientated database called ClickHouseDB to host very large time series data tables. This class of database outperforms traditional row-orientated databases in both speed and storage.

The project aim is to create an interactive demonstrator running on Rockstone Data’s website clearly demonstrating these benefits vs ‘traditional’ or row-orientated databases.

### List of contents

[1 Introduction 4](#_Toc164955145)

[1.1 Purpose of the document 4](#_Toc164955146)

[1.2 Scope of the document 4](#_Toc164955147)

[1.3 Definitions, acronyms and abbreviations 4](#_Toc164955148)

[1.4 Conventions 5](#_Toc164955149)

[1.5 Stakeholders 6](#_Toc164955150)

[2 Architectural Design 8](#_Toc164955151)

[2.1 System Overview 8](#_Toc164955152)

[2.2 Design Method 8](#_Toc164955153)

[2.3 Component Overview 8](#_Toc164955154)

[2.4 Deployment View 9](#_Toc164955155)

[2.5 Scenarios 9](#_Toc164955156)

[2.6 Development Environment 9](#_Toc164955157)

[2.7 Testing Environment 9](#_Toc164955158)

[3 Detailed Design 11](#_Toc164955159)

[3.1 User Interface 11](#_Toc164955160)

[3.2 Class Model 11](#_Toc164955161)

[3.3 Database Design 13](#_Toc164955162)

[4 References / Bibliography 14](#_Toc164955163)

### Introduction

#### Purpose of the document

This document describes the architectural design and detailed design for the project.

This Design Document is to be a living document for the duration of the project. It will be updated and expanded upon, at least at the start of each stage or iteration, and as required throughout the project’s software development lifecycle.

### Scope of the document

This document covers all aspects of the software design.

### Definitions, acronyms and abbreviations

The Project Plan Document [1] contains the definitions, acronyms and abbreviations for this project. The glossary will also be the data dictionary, so there is a common understanding of terms and consistent naming of objects within the development.

Project Initiation Document: Summary of important information about the project, including its context, scope, milestones, objectives and requirements.

Proof of Concept: Gathering evidence to gauge the feasibility of a project.

ClickHouseDB: An open-source database which is often used for queries involving real-time and historical data.

Postgres (PostgreSQL): An open-source relational database often used for data storage for web, mobile, geospatial and analytics applications.

TimescaleDB: A database engine that uses Postgres for applications requiring time series, vector, events and analytics data.

MongoDB: A document database often used for scaling throughout application development.

Docker / Docker Compose: A tool used to run multi-container applications. Each container provides different back-end functionality for the application.

DBMS (Database Management System): Software systems used to store and fetch data.

AWS (Amazon Web Services): An infrastructure platform which provides on-demand cloud computing and APIs for deploying applications.

EC2 (Amazon Elastic Compute Cloud): A compute platform used for deploying applications on the cloud.

OLAP (Online Analytical Processing): A form of computing whereby the user can extract and analyse data from different perspectives.

OLTP (Online Transaction Processing): A form of data processing whereby the user can execute many transactions at the same time.

Snowflake: Software as a service which can be used for data warehouses, lakes, sharing of data, data science and engineering, and data application management.

Data Dog: Software as a service which can be used for monitoring of servers, databases, tools and services.

DigitalOcean: Software and infrastructure as a service which can be used for deploying web applications.

ArcticDB: An embedded and serverless database used for storing and processing Pandas DataFrames.

### Conventions

The Standards Document [2] contains naming conventions and terminology.

Branch Naming Conventions:

The GitHub development flow is used so as to allow individuals to be able to create a branch for issues they are currently working on.

The branches are named like the following example: nt\_123\_short\_desc.

Breaking this down,

* nt will be the developer's initials.
* 123 will be the Issue Number.
* There will then be a one- or two-word description of the issue.

Joel Test:

The project will try and adhere to the 'Joel Test' of good software practice.

The Joel Test is a very simple and quick test that rates the quality of your software team. Rather than including open-ended responses, this test has 12 yes-or-no questions that determine whether your programming team is up to par. A score of 12 is perfect. 11 is considered tolerable, and 10 or below is unacceptable. These 12 questions are as follows:

1. Do you use source control?

2. Can you make a build in one step?

3. Do you make daily builds?

4. Do you have a bug database?

5. Do you fix bugs before writing new code?

6. Do you have an up-to-date schedule?

7. Do you have a spec?

8. Do programmers have quiet working conditions?

9. Do you use the best tools money can buy?

10. Do you have testers?

11. Do new candidates write code during their interview?

12. Do you do hallway usability testing?

Coding Conventions:

The coding conventions for this project will follow the PEP 8 Conventions - This is documentation that provides guidelines and best practices on how to write Python code. This can be viewed here: PEP 8 Documentation.

When coding standards are properly defined and implemented, developers, even those who have just joined the team, can easily find their way around the code base. Ideally, wanting our source code to look like a single developer has written, debugged, and maintained it.

Code Layout

File Header - File headers will be at the top of every file and include the following information:

* FileName - Will be the name of the file.
* FileType - Will be the type of file i.e., py (python), cs (c#), js (JavaScript)
* Created By - Who it was created by. Formatted as Surname, First Name
* Created On - The date and time it was created on in a dd/mm/yyyy hh:mm:ss AM/PM format
* Last Modified - The date and time it was last modified on in a dd/mm/yyyy hh:mm:ss AM/PM format
* Description - Brief description on what the file does

### Stakeholders

The list of stakeholders is contained within the Project Plan [1].

|  |  |  |
| --- | --- | --- |
| Role | Stakeholder Name | Details |
| Project Sponsor | Nick Thorne | Director and Consultant Software Engineer at Rockstone Data  Primary contact for the project |
| Support Tutor | Martin Reid | Lecturer and Apprenticeship Advisor at Solent University  Must be provided with project progress update after each Sprint |

### Architectural Design

#### System Overview

The Requirements Document contains information on an overview of the final product [3].

The system will be a Dashboard web application hosted on Docker containers, featuring a line plot depicting comparison data from ClickHouseDB. Our group will design, develop, test and deploy the software according to the requested requirements supplied by the project sponsor, Rockstone Data.

#### Design Method

The information for the design method can be found within the Project Plan [1].

Python 3.11 was utilised for the Application back-end and front-end code.

An agile methodology utilising three iterative sprints was incorporated.

It will adopt some elements from Scrum, including the roles of Product Owner and Scrum Master, and the existence of Scrum ceremonies such as Sprint Planning, Sprint Review and Sprint Retrospective.

GitHub Project Board, GitHub Issues Board and Microsoft Word were used for Software Project Management

A GitHub Code Repository was used for Version Control

Microsoft Word and Excel were used for tracking Requirements.

Microsoft Word and Wireframe.cc were used for Analysis and Design

Implementation was ensured via the use of; Microsoft Visual Studio Code, Streamlit OR Plotly Dash, Python pip, Python virtual environments (venv), ClickHouseDB, TimescaleDB, PostgreSQL, MongoDB, Snowflake and Data Dog

Testing was done via Microsoft Visual Studio Code

Deployment carried out via Docker Desktop and Docker Compose

Support and Maintenance was carried out through GitHub Wiki Pages

#### Component Overview

The system will consist of 4 databases (PostgresqlDB, ClickhouseDB, ArcticDB, TimeScaleDB)

Data from the databases will be fetched and plotted on a scaler vector.

A timer will record how long the fetching process takes for each database.

The data range will be inputted by a user and the graph will reflect that range.

A downsampling option is available and will be utilised when selected.

The number of rows and disk storage required for each database will be recorded and displayed along with the timer.

A diagram of a diagram

Description automatically generated with medium confidence

#### Deployment View

Deployment is covered within the Project Plan [1].

Deployment will be completed via the use of Docker Desktop and Docker Compose and once deployed will run on all devices that can utilise streamlit.

It may later be deployed using Amazon Web Services EC2 or DigitalOcean

#### Scenarios

The system is able to display the differences between ClickHouseDB’s column-orientated format and a traditional database’s row-orientated format using a line plot. In addition, the application can also display this information through geo-heatmaps or data analysis using Snowflake and Data Dog.

Large data sets (for example stock market data) are fetched and then displayed as a scalar plot, this information has the option to be downsampled and can be displayed over specific time instances (24 hours, 7 days, 1 year). This is carried out through various databases which are then compared through key metrics such as ‘Time Elapsed’, ‘Disk Storage Used’ and ‘Number of Rows Required’, this information is then displayed to the user. Stock Market Data is a suitable data set for this due to its large and continuous nature.

#### Development Environment

Environment information can be found within the Project Plan [1].

The final product shall be able to run according to the advised minimum specifications for running Python 3.11, therefore, the product will not be able to run on Windows 7 or earlier. It will run better on computers with larger RAM and processing power due to the Machine Learning techniques used.

Once deployed, the application will run on all devices which can run Streamlit.

#### Testing Environment

Test Environment can be found within the Project Plan [1].

Test Progress [5] demonstrates implementation of testing.

1 - Reviews and Inspections

All code must be placed into a pull request on GitHub before being merged into the master branch. The pull request must be reviewed and approved by at least one other team member. Commit messages that summarise the code changes must be added to branch commits and to the resulting pull request.

2 - Testing

The software development team will perform unit testing using pytest, system testing and acceptance testing. All software components must undergo unit testing, and the test results, harnesses and input data must be documented in the test document. The test document will also include the system and acceptance test strategies and test scripts.

3 - Lessons Learned

Lessons learned will be discussed throughout the project and documented as part of the Sprint Retrospective.

The Requirements for testing are found in the Requirements Document [3].

Test data shall contain visible results on a 24-hour basis

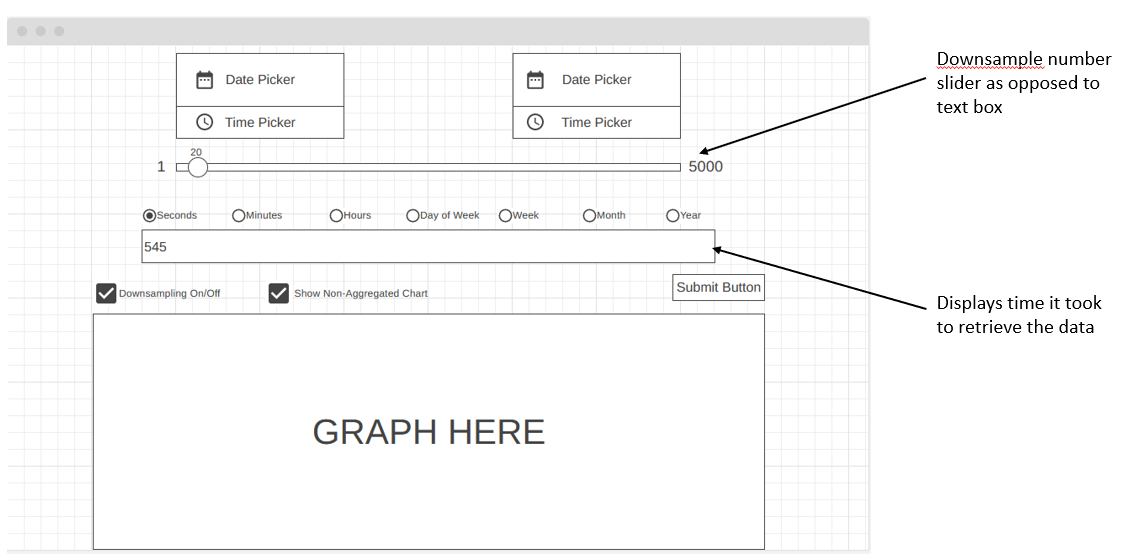
Test data shall contain visible results on a 1-month basis

Test data shall contain visible results on a 1-year basis

### Detailed Design

#### User Interface

The User Interface Requirements can be found within the Requirements Document [3).



- The application shall be a multi-page Dashboard web application of at least 2 pages.

- The user interface shall display a line plot of a scalar value over time.

- The user interface shall display a line plot of a start / end datetime picker.

- The user interface shall display a line plot of a database source picker.

- The user interface shall have a ‘Submit’ button which fetches the data for the line plot.

- The user interface shall have a ‘Downsampling On/Off’ toggle.

- The user interface shall have a ‘Downsampling’ text entry form.

- The user interface may have a multidimensional class of data displayed as a geo-heatmap or spectrograph or surface plot.

- The graphs may have additional animations or orientation controls.

- The multidimensional data may be shown below the line chart.

- The user interface may compare costs using Snowflake or Data Dog.

- The application may benchmark writing data to large tables.

- The application may benchmark the ArcticDB time-series collections

#### Class Model

Sequence diagram:

A diagram of a data flow

Description automatically generated

Class diagram:

A diagram of a graph

Description automatically generated

#### Database Design

Database Benchmarking [4] contains details of the database design.

In the Design phase, the organisation will determine what aspects of the database need to be measured - such as performance, scalability, availability, elasticity or cost – and how to

measure them. After this, the organisation will choose an appropriate benchmark and workload.

Lastly, test runs of the benchmarking process will occur in several iterations to obtain an idea on what reliable results should look like.

Databases used:

ArcticDB

ClickhouseDB

PostgresqlDB

TimescaleDB

The databases are not relational, they are accessed via fetching before the data is displayed on a graph. The relationship between the database and the graph is 1-1.

A diagram of a diagram

Description automatically generated

### Design Document References / Bibliography

docs/RequirementsDocument.docx

1. GitHub. (n.d.). db\_bench/docs/Standards.md at master · NickThorne123/db\_bench. <https://github.com/NickThorne123/db_bench/blob/master/docs/RequirementsDocument.docx>
2. GitHub. (n.d.). db\_bench/docs/Standards.md at master · NickThorne123/db\_bench. <https://github.com/NickThorne123/db_bench/blob/master/docs/Standards.md>
3. GitHub. (n.d.). db\_bench/docs/Standards.md at master · NickThorne123/db\_bench. [https://github.com/NickThorne123/db\_bench/blob/master/docs/RequirementsDocument.docx](https://github.com/NickThorne123/db_bench/blob/master/docs/RequirementsDocument.docx%20)
4. GitHub. (n.d.). db\_bench/docs/Standards.md at master · NickThorne123/db\_bench. [https://github.com/NickThorne123/db\_bench/blob/master/docs/Research/Database Benchmarking.docx](https://github.com/NickThorne123/db_bench/blob/master/docs/Standards.md)
5. GitHub. (n.d.). db\_bench/docs/Standards.md at master · NickThorne123/db\_bench. <https://github.com/NickThorne123/db_bench/blob/master/tests/test_postgress.py>

## Appendix B – User Guide

### List of contents

[**1** **Setup [1]** 4](#_Toc165300003)

[1.1 Installing and Using this Repo 4](#_Toc165300004)

[1.2 Configuring ClickHouseDB 5](#_Toc165300005)

[1.3 Configuring PostgreSQL 6](#_Toc165300006)

[1.4 Configuring TimescaleDB 7](#_Toc165300007)

[1.5 Configuring ArcticDB 8](#_Toc165300008)

[1.6 Troubleshooting 8](#_Toc165300009)

[2 Page Interface 9](#_Toc165300010)

[2.1 Home Page 9](#_Toc165300011)

[2.2 Database Pages 9](#_Toc165300012)

### Setup [1]

### Installing and Using this Repo

This This project has four main componenents 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](https://code.visualstudio.com/docs/languages/python#_debugging) 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](https://docs.docker.com/desktop/install/mac-install/) on your machine.

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](https://dbeaver.io/download/). 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 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 toDate(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

);

### 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;

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

);

### 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.eu-west-2.amazonaws.com:<bucket\_name>?aws\_auth=true' to ARCTIC\_URL.

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.

A close-up of a benchmarking

Description automatically generated

Navigation Panel

#### Database Pages

Each database page will consist of:

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

A screenshot of a calendar

Description automatically generated

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

Off:

A screenshot of a computer

Description automatically generated

On:

A screenshot of a computer

Description automatically generated

* The downsampling count text entry

A screenshot of a computer

Description automatically generated

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

A screenshot of a computer

Description automatically generated

Once the submit button is 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 GB of disk storage per million rows.
* A graph of the fetched data.

A screenshot of a computer

Description automatically generated

A screen shot of a graph

Description automatically generated

## A screen shot of a data chart Description automatically generatedA screenshot of a computer Description automatically generatedAppendix C – ClickhouseDB Stats

## A screenshot of a computer Description automatically generatedA screenshot of a computer Description automatically generatedAppendix D – PostgreSQL Stats

A screenshot of a data chart

Description automatically generated

## A screenshot of a computer Description automatically generatedAppendix E – TimeseriesDB Stats

A screen shot of a data chart

Description automatically generatedA screenshot of a computer

Description automatically generated

## A screenshot of a computer Description automatically generatedAppendix F – ArcticDB Stats

A screen shot of a graph

Description automatically generatedA screenshot of a computer

Description automatically generated

## A screenshot of a computer Description automatically generatedAppendix G – Sprint Overview

A white and black grid with black text

Description automatically generated with medium confidence

## Appendix H – Test Plan Document

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST TITLE** |  |  |  |
| COM617 DB Benchmark Test Plan |  |  |  |
| **TEST DESCRIPTION** |  |  |  |
| Unit tests to be created alongside the code and run via Github Actions |  |  |
| **TEST DESCRIPTION** | **TEST DEPENDENCIES** | **TEST CONDITIONS** | **TEST CONTROL** |
| Unit tests to be created alongside the code and run via Github Actions after each push to the repository. This test plan should make up a part of the CI/CD Pipeline and should be updated accordingly when features are added. The tests can also be run manually by the user using Pytest in the IDE. | The dependencies will vary as the software becomes more complex as some tests will require different databases to be run. | Tests are to be run automatically through Github actions. Tests should also be able to be run manually on branch using Pytest. | Test to be run automatically after each push by Github so should be in a controlled environment. To test manually, first close application then run through. |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **STEP ID** | **STEP DESCRIPTION** | **TEST DATE** | **EXPECTED RESULTS** | **ACTUAL RESULTS** | **PASS / FAIL** | **ADDITIONAL NOTES** |
| 1 | A test to check the Home Page opens | 03/24/2024 | Test Pass- The home page can be opened | Test Pass- The home page can be opened | PASS |  |
| 2 | Downsampling Toggle can be selected for DB1 – (Clickhouse) | 03/24/2024 | Test Pass – The downsampling toggles can be selected for each database type | Test Pass – The downsampling toggles can be selected for each database type | PASS |  |
| 3 | Increment the Downsample Number Input for each Database type – DB 1 (Clickhouse) | 03/24/2024 | Test Pass – The downsampling value is incremented by 1 to a value of 6 (min is 5) | Test Pass – The downsampling value is incremented by 1 to a value of 6 (min is 5) | PASS |  |
| 4 | Set the Downsample Number Input for each Database type – DB 1 (Clickhouse) | 03/24/2024 | Test Pass – The downsampling value is set to a value of 5000 | Test Pass – The downsampling value is set to a value of 5000 | PASS |  |
| 5 | Change the start date of each Database type – DB 1 (Clickhouse) | 03/24/2024 | Test Pass – The start date can be changed | Test Pass – The start date can be changed | PASS |  |
| 6 | Change the end date of each Database type – DB 1 (Clickhouse) | 03/24/2024 | Test Pass – The end date can be changed | Test Pass – The end date can be changed | PASS |  |
| 7 | Change the start time of each Database type – DB 1 (Clickhouse) | 03/24/2024 | Test Pass – The start time can be changed | Test Pass – The start time can be changed | PASS |  |
| 8 | Change the end time of each Database type – DB 1 (Clickhouse) | 03/24/2024 | Test Pass – The end time can be changed | Test Pass – The end time can be changed | PASS |  |
| 9 | Submit Button can be selected and data ingressed for each Database type – DB 1 (Clickhouse) | 03/24/2024 | Test Pass – The submit button can be selected and the data is imported | Test Expected Fail (Due to pytest timeout) - The submit button can be selected and the data is imported | XFAIL | Test times out after 3 seconds, set as XFAIL when it times out passes as it shows data is being imported |
| 10 | Downsampling Toggle can be selected for DB 2 – (PostgreSQL) | 03/24/2024 | Test Pass – The downsampling toggles can be selected for each database type | Test Pass – The downsampling toggles can be selected for each database type | PASS |  |
| 11 | Increment the Downsample Number Input for each Database type – DB 2 (PostgreSQL) | 03/24/2024 | Test Pass – The downsampling value is incremented by 1 to a value of 6 (min is 5) | Test Pass – The downsampling value is incremented by 1 to a value of 6 (min is 5) | PASS |  |
| 12 | Set the Downsample Number Input for each Database type – DB 2 (PostgreSQL) | 03/24/2024 | Test Pass – The downsampling value is set to a value of 5000 | Test Pass – The downsampling value is set to a value of 5000 | PASS |  |
| 13 | Change the start date of each Database type – DB 2 (PostgreSQL) | 03/24/2024 | Test Pass – The start date can be changed | Test Pass – The start date can be changed | PASS |  |
| 14 | Change the end date of each Database type – DB 2 (PostgreSQL) | 03/24/2024 | Test Pass – The end date can be changed | Test Pass – The end date can be changed | PASS |  |
| 15 | Change the start time of each Database type – DB 2 (PostgreSQL) | 03/24/2024 | Test Pass – The start time can be changed | Test Pass – The start time can be changed | PASS |  |
| 16 | Change the end time of each Database type – DB 2 (PostgreSQL) | 03/24/2024 | Test Pass – The end time can be changed | Test Pass – The end time can be changed | PASS |  |
| 17 | Submit Button can be selected and data ingressed for each Database type – DB 2 (PostgreSQL) | 03/24/2024 | Test Pass – The submit button can be selected and the data is imported | Test Expected Fail (Due to pytest timeout) - The submit button can be selected and the data is imported | XFAIL | Test times out after 3 seconds, set as XFAIL when it times out passes as it shows data is being imported |
| 18 | Downsampling Toggle can be selected for DB 3 – (TimescaleDB) | 03/24/2024 | Test Pass – The downsampling toggles can be selected for each database type | Test Pass – The downsampling toggles can be selected for each database type | PASS |  |
| 19 | Increment the Downsample Number Input for each Database type – DB 3 (TimescaleDB) | 03/24/2024 | Test Pass – The downsampling value is incremented by 1 to a value of 6 (min is 5) | Test Pass – The downsampling value is incremented by 1 to a value of 6 (min is 5) | PASS |  |
| 20 | Set the Downsample Number Input for each Database type – DB 3 (TimescaleDB) | 03/24/2024 | Test Pass – The downsampling value is set to a value of 5000 | Test Pass – The downsampling value is set to a value of 5000 | PASS |  |
| 21 | Change the start date of each Database type – DB 3 (TimescaleDB) | 03/24/2024 | Test Pass – The start date can be changed | Test Pass – The start date can be changed | PASS |  |
| 22 | Change the end date of each Database type – DB 3 (TimescaleDB) | 03/24/2024 | Test Pass – The end date can be changed | Test Pass – The end date can be changed | PASS |  |
| 23 | Change the start time of each Database type – DB 3 (TimescaleDB) | 03/24/2024 | Test Pass – The start time can be changed | Test Pass – The start time can be changed | PASS |  |
| 24 | Change the end time of each Database type – DB 3 (TimescaleDB) | 03/24/2024 | Test Pass – The end time can be changed | Test Pass – The end time can be changed | PASS |  |
| 25 | Submit Button can be selected and data ingressed for each Database type – DB 3 (TimescaleDB) | 03/24/2024 | Test Pass – The submit button can be selected and the data is imported | Test Pass – The submit button can be selected | PASS | Currently Passes as Timescale has not got data yet |
| 26 | Downsampling Toggle can be selected for DB 4 – (ArcticDB) | 03/24/2024 | Test Pass – The downsampling toggles can be selected for each database type | Test Pass – The downsampling toggles can be selected for each database type | PASS |  |
| 27 | Increment the Downsample Number Input for each Database type – DB 4 (ArcticDB) | 03/24/2024 | Test Pass – The downsampling value is incremented by 1 to a value of 6 (min is 5) | Test Pass – The downsampling value is incremented by 1 to a value of 6 (min is 5) | PASS |  |
| 28 | Set the Downsample Number Input for each Database type – DB 4 (ArcticDB) | 03/24/2024 | Test Pass – The downsampling value is set to a value of 5000 | Test Pass – The downsampling value is set to a value of 5000 | PASS |  |
| 29 | Change the start date of each Database type – DB 4 (ArcticDB) | 03/24/2024 | Test Pass – The start date can be changed | Test Pass – The start date can be changed | PASS |  |
| 30 | Change the end date of each Database type – DB 4 (ArcticDB) | 03/24/2024 | Test Pass – The end date can be changed | Test Pass – The end date can be changed | PASS |  |
| 31 | Change the start time of each Database type – DB 4 (ArcticDB) | 03/24/2024 | Test Pass – The start time can be changed | Test Pass – The start time can be changed | PASS |  |
| 32 | Change the end time of each Database type – DB 4 (ArcticDB) | 03/24/2024 | Test Pass – The end time can be changed | Test Pass – The end time can be changed | PASS |  |
| 33 | Submit Button can be selected and data ingressed for each Database type – DB 4 (ArcticDB) | 03/24/2024 | Test Pass – The submit button can be selected and the data is imported | Test Expected Fail (Due to pytest timeout) - The submit button can be selected and the data is imported | PASS | Test times out after 3 seconds, set as XFAIL when it times out passes as it shows data is being imported |
| 34 | All values can be edited and stay the selected values when the submit button is selected – DB 1 (Clickhouse) | 03/24/2024 | Test Pass – The submit button can be selected and all the changed fields remain changed | Test Expected Fail (Due to pytest timeout) - The submit button can be selected but times out due to data import | XFAIL | Test times out after 3 seconds, set as XFAIL when it times out passes as it shows data is being imported |
| 35 | All values can be edited and stay the selected values when the submit button is selected – DB 2 (PostgreSQL) | 03/24/2024 | Test Pass – The submit button can be selected and all the changed fields remain changed | Test Expected Fail (Due to pytest timeout) - The submit button can be selected but times out due to data import | XFAIL | Test times out after 3 seconds, set as XFAIL when it times out passes as it shows data is being imported |
| 36 | All values can be edited and stay the selected values when the submit button is selected – DB 3 (TimescaleDB) | 03/24/2024 | Test Pass – The submit button can be selected and all the changed fields remain changed | Test Pass – The submit button can be selected | PASS | Currently Passes as Timescale has not got data yet |
| 37 | All values can be edited and stay the selected values when the submit button is selected – DB 4 (ArcticDB) | 03/24/2024 | Test Pass – The submit button can be selected and all the changed fields remain changed | Test Expected Fail (Due to pytest timeout) - The submit button can be selected but times out due to data import | XFAIL | Test times out after 3 seconds, set as XFAIL when it times out passes as it shows data is being imported |
| 38 | Change the start date of each Database type (Writing Data Benchmark) – DB 1 (ClickhouseDB) | 04/06/2024 | Test Pass – The start date can be changed | Test Pass – The start date can be changed | PASS |  |
| 39 | Change the end date of each Database type (Writing Data Benchmark) – DB 1 (ClickhouseDB) | 04/06/2024 | Test Pass – The end date can be changed | Test Pass – The end date can be changed | PASS |  |
| 40 | Change the start time of each Database type (Writing Data Benchmark) – DB 1 (ClickhouseDB) | 04/06/2024 | Test Pass – The start time can be changed | Test Pass – The start time can be changed | PASS |  |
| 41 | Change the end time of each Database type (Writing Data Benchmark) – DB 1 (ClickhouseDB) | 04/06/2024 | Test Pass – The end time can be changed | Test Pass – The end time can be changed | PASS |  |
| 42 | Submit Button can be selected and data ingressed for each Database type (Writing Data Benchmark) – DB 1 (ClickhouseDB) | 04/06/2024 | Test Pass – The submit button can be selected and the data is imported | Test Expected Fail (Due to pytest timeout) - The submit button can be selected but times out due to data import | XFAIL | Test times out after 3 seconds, set as XFAIL when it times out passes as it shows data is being imported |
| 43 | Change the start date of each Database type (Writing Data Benchmark) – DB 2 (PostgresSQL) | 04/06/2024 | Test Pass – The start date can be changed | Test Pass – The start date can be changed | PASS |  |
| 44 | Change the end date of each Database type (Writing Data Benchmark) – DB 2 (PostgresSQL) | 04/06/2024 | Test Pass – The end date can be changed | Test Pass – The end date can be changed | PASS |  |
| 45 | Change the start time of each Database type (Writing Data Benchmark) – DB 2 (PostgresSQL) | 04/06/2024 | Test Pass – The start time can be changed | Test Pass – The start time can be changed | PASS |  |
| 46 | Change the end time of each Database type (Writing Data Benchmark) – DB 2 (PostgresSQL) | 04/06/2024 | Test Pass – The end time can be changed | Test Pass – The end time can be changed | PASS |  |
| 47 | Submit Button can be selected and data ingressed for each Database type (Writing Data Benchmark) – DB 2 (PostgresSQL) | 04/06/2024 | Test Pass – The submit button can be selected and the data is imported | Test Expected Fail (Due to pytest timeout) - The submit button can be selected but times out due to data import | XFAIL | Test times out after 3 seconds, set as XFAIL when it times out passes as it shows data is being imported |
| 48 | Change the start date of each Database type (Writing Data Benchmark) – DB 3 (TimescaleDB) | 04/19/2024 | Test Pass – The start date can be changed | Test Pass – The start date can be changed | PASS |  |
| 49 | Change the end date of each Database type (Writing Data Benchmark) – DB 3 (TimescaleDB) | 04/19/2024 | Test Pass – The end date can be changed | Test Pass – The end date can be changed | PASS |  |
| 50 | Change the start time of each Database type (Writing Data Benchmark) – DB 3 (TimescaleDB) | 04/19/2024 | Test Pass – The start time can be changed | Test Pass – The start time can be changed | PASS |  |
| 51 | Change the end time of each Database type (Writing Data Benchmark) – DB 3 (TimescaleDB) | 04/19/2024 | Test Pass – The end time can be changed | Test Pass – The end time can be changed | PASS |  |
| 52 | Submit Button can be selected and data ingressed for each Database type (Writing Data Benchmark) – DB 3 (TimescaleDB) | 04/19/2024 | Test Pass – The submit button can be selected and the data is imported | Test Expected Fail (Due to pytest timeout) - The submit button can be selected but times out due to data import | XFAIL | Test times out after 3 seconds, set as XFAIL when it times out passes as it shows data is being imported |
| 53 | Change the start date of each Database type (Writing Data Benchmark) – DB 4 (ArcticDB) | 04/06/2024 | Test Pass – The start date can be changed | Test Pass – The end date can be changed | PASS |  |
| 54 | Change the end date of each Database type (Writing Data Benchmark) – DB 4 (ArcticDB) | 04/06/2024 | Test Pass – The end date can be changed | Test Pass – The start time can be changed | PASS |  |
| 55 | Change the start time of each Database type (Writing Data Benchmark) – DB 4 (ArcticDB) | 04/06/2024 | Test Pass – The start time can be changed | Test Pass – The end time can be changed | PASS |  |
| 56 | Change the end time of each Database type (Writing Data Benchmark) – DB 4 (ArcticDB) | 04/06/2024 | Test Pass – The end time can be changed | Test Pass – The start date can be changed | PASS |  |
| 57 | Submit Button can be selected and data ingressed for each Database type (Writing Data Benchmark) – DB 4 (ArcticDB) | 04/06/2024 | Test Pass – The submit button can be selected and the data is imported | Test Expected Fail (Due to pytest timeout) - The submit button can be selected but times out due to data import | XFAIL | Test times out after 3 seconds, set as XFAIL when it times out passes as it shows data is being imported |
| 58 | All values can be edited and stay the selected values when the submit button is selected (Writing Data Benchmark) – DB 1 (Clickhouse) | 04/06/2024 | Test Pass – The submit button can be selected and all the changed fields remain changed | Test Expected Fail (Due to pytest timeout) - The submit button can be selected but times out due to data import | XFAIL | Test times out after 3 seconds, set as XFAIL when it times out passes as it shows data is being imported |
| 59 | All values can be edited and stay the selected values when the submit button is selected (Writing Data Benchmark) – DB 2 (PostgreSQL) | 04/06/2024 | Test Pass – The submit button can be selected and all the changed fields remain changed | Test Expected Fail (Due to pytest timeout) - The submit button can be selected but times out due to data import | XFAIL | Test times out after 3 seconds, set as XFAIL when it times out passes as it shows data is being imported |
| 60 | All values can be edited and stay the selected values when the submit button is selected (Writing Data Benchmark) – DB 3 (TimescaleDB) | 04/19/2024 | Test Pass – The submit button can be selected and all the changed fields remain changed | Test Expected Fail (Due to pytest timeout) - The submit button can be selected but times out due to data import | XFAIL | Test times out after 3 seconds, set as XFAIL when it times out passes as it shows data is being imported |
| 61 | All values can be edited and stay the selected values when the submit button is selected (Writing Data Benchmark) – DB 4 (ArcticDB) | 04/06/2024 | Test Pass – The submit button can be selected and all the changed fields remain changed | Test Expected Fail (Due to pytest timeout) - The submit button can be selected but times out due to data import | XFAIL | Test times out after 3 seconds, set as XFAIL when it times out passes as it shows data is being imported |

## **A screenshot of a computer Description automatically generated**Appendix I – Project Requirements Table