< >

COM617 Design Document – Database Benchmarking

Design Document (Template issue: 1.0)

Enter Author Name: Luke Wood, Iona Pitt, Josh Clarke, Kyle Roberts, Danny Agha

Enter Report Date: 15/04/24

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

1.1 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.

1.2 Scope of the document

This document covers all aspects of the software design.

1.3 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.

1.4 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

1.5 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

2 Architectural Design

2.1 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.

2.2 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

2.3 Component Overview

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

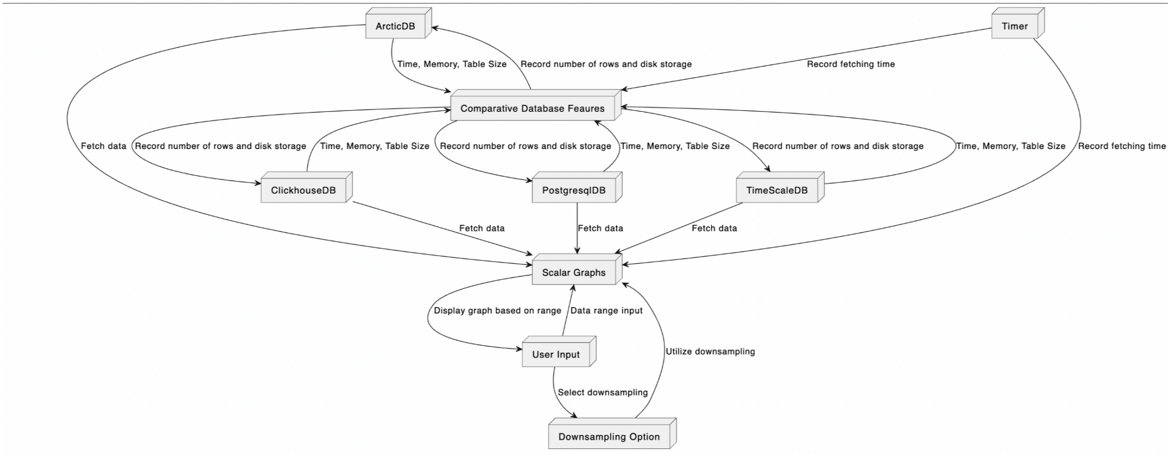
Data from the databases will be fetched and plotted on a scaler vector.

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.



2.4 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

2.5 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.

2.6 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.

2.7 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

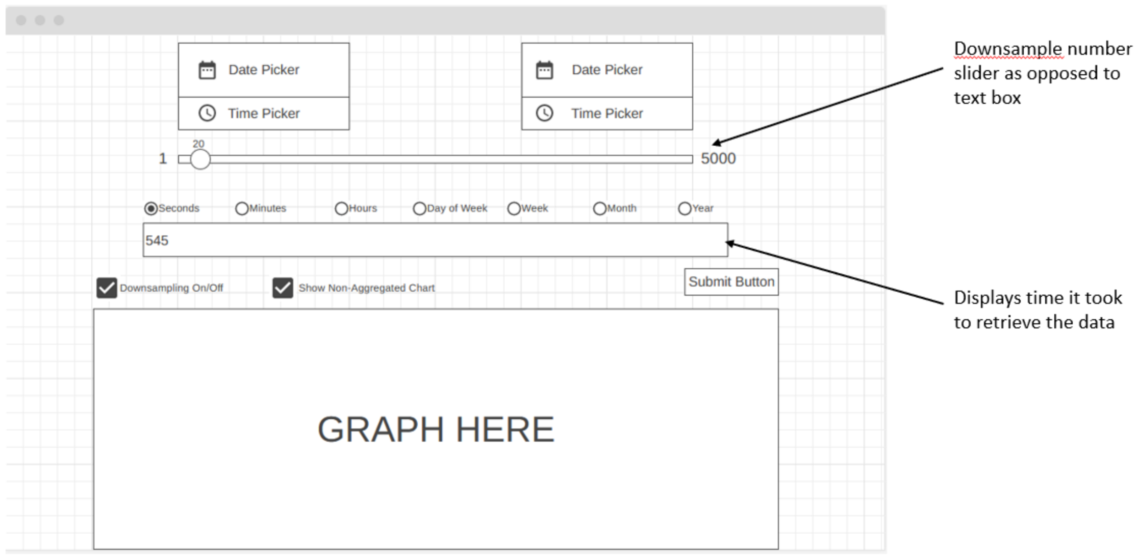
Test data shall contain visible results on a 1-month basis

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

3 Detailed Design

3.1 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.

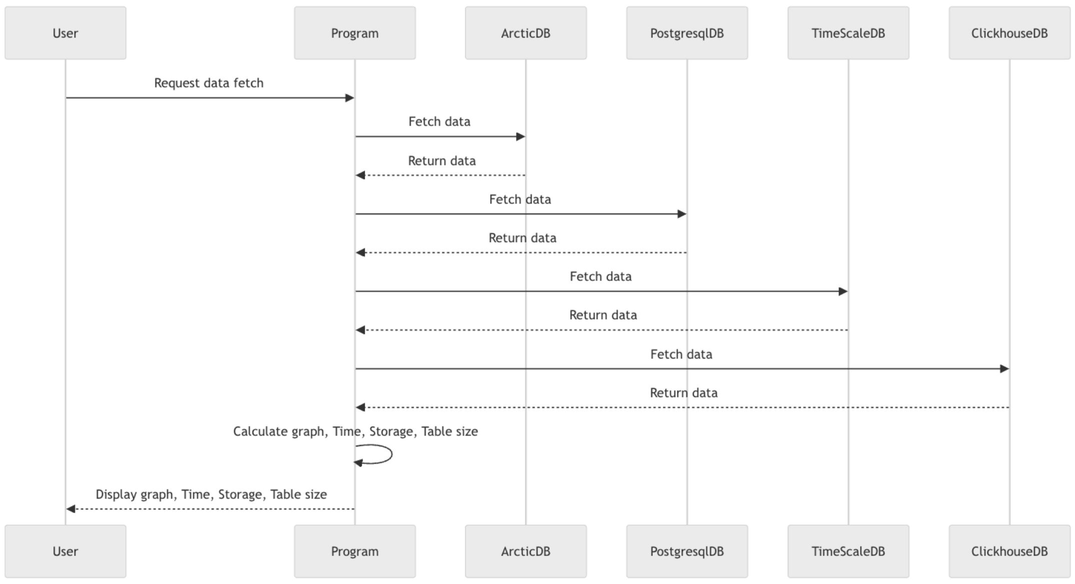
- The user interface may compare costs using Snowflake or Data Dog.

- The application may benchmark writing data to large tables.

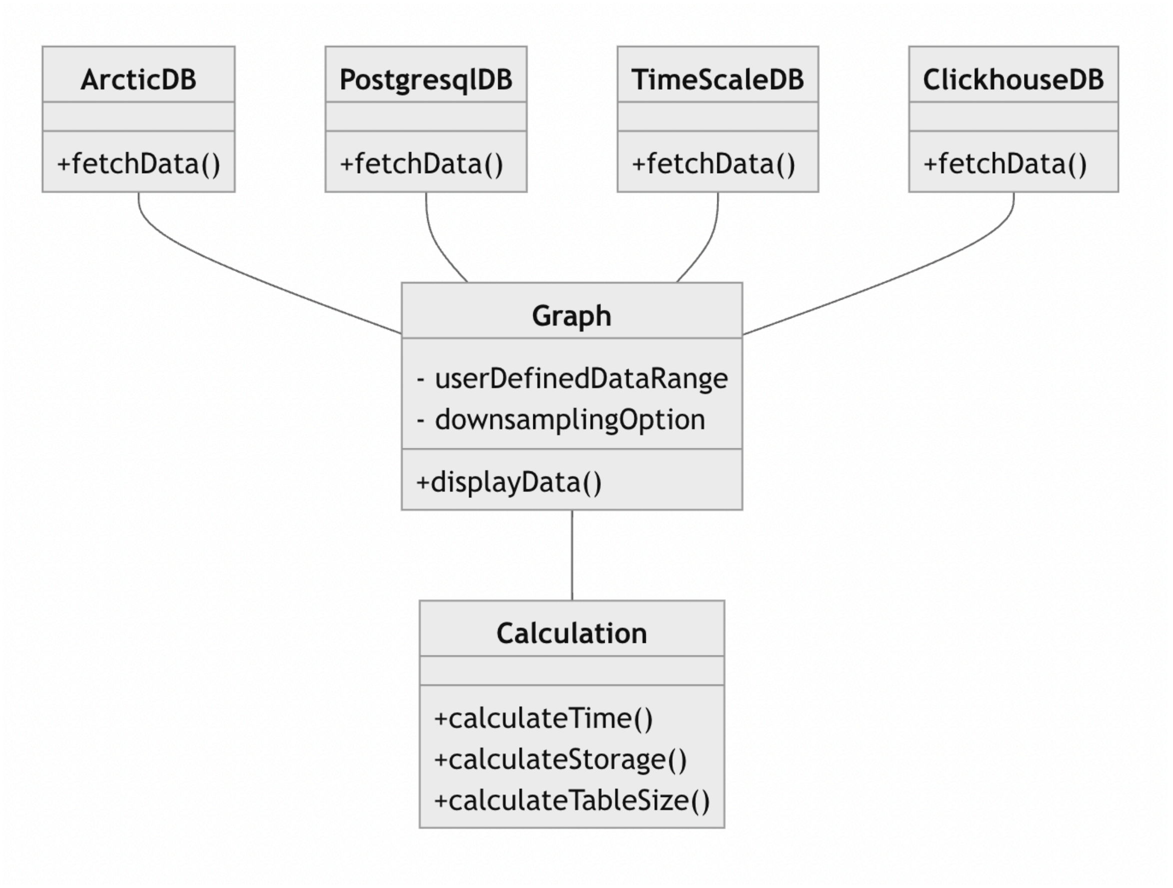
- The application may benchmark the ArcticDB time-series collections

3.2 Class Model

Sequence diagram:



Class diagram:



3.3 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:

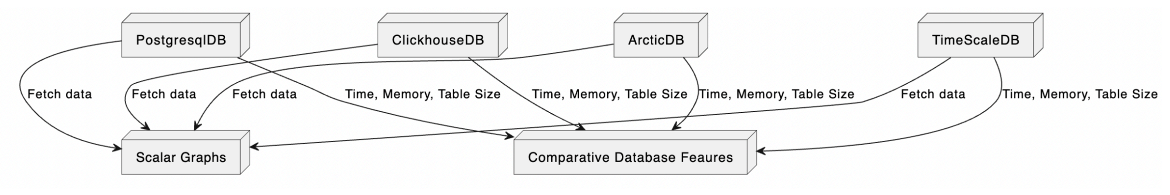
ArcticDB

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.



4 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

[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

[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

Blank page