SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Visualisation and statistical analysis of performance measurements of database systems

Julian Macias De La Rosa

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Visualisation and statistical analysis of performance measurements of database systems

Visualisierung und statistische Aufbereitung von Performance Messungen von Datenbanksystemen

Author:	Julian Macias De La Rosa
Supervisor:	Prof. Thomas Neumann
Advisor:	Maximilian Bandle
Submission Date:	20.10.2023

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 20.10.2023 Julian Macias De La Rosa

Acknowledgments

Abstract

Contents

1 Introduction

Interactive performance visualisation is a powerfull skill and plays a vital role for the demonstration of meaningful data insights in the context of performance measurements. Our goal is to use this powerfull skill properly to enable potential optimization possibilities for compiling database systems.

1.1 Motivation

1.2 Technical Background

1.2.1 React

1.2.2 Redux

1.2.3 React Sweet State//

1.2.4 Plotly

1.2.5 React-Flow

1.2.6 Material UI

1.3 Existing Visualisation of Performance Data of Umbra //PDF

1.4 Research objectives

1.5 Scope and contribution of the thesis

1.6 Thesis structure

2 Related Work

In this chapter, we give an overview about the existing work in the domain of the visualisation of database performance profiling. We will investigate the importance of optimizing query executions in database systems and the role of visualisations in identifying potential improvements. As performant measurement and analysis play a crucial role in developing and optimizing database systems, it is essential to examine the state-of-the-art techniques and tools that have been used in this domain. We will also cover a visualisation tool closely associated with this thesis, as its key feature is integrated into the Benchy Viewer.

2.1 Performance Visualisation

Sektion eher in Background Todo: Was es alles in dieser Domain gibt. Was effektiv ist und wir benutzen. Was wir nicht benutzen mit Begründung.

2.2 Database Performance Profiling

Performance profiling in database systems is crucial for optimizing their execution regarding achieving optimal hardware utilization and query efficiency. Profiling the performance of database systems involves collecting and analyzing various performance metrics during query execution.
Besides profilers presenting results at the instruction and function granularity, a paper on "Profiling Dataflow Systems on Multiple Abstraction Levels" [Bei+21] proposes a solution that tracks the code generation process and aggregates profiling data to higher abstraction levels. This approach helps bridging the semantic gap between low-level profiles and high-level constructs, making it easier for developers to interpret profiling results and identify bottlenecks and hotspots in the system. The paper introduces the concept of Tailored Profiling, which extends the compilation steps to annotate the generated code with metadata. This enables the mapping of profiling results back to desired abstraction levels and provides more understandable profiling data. Building

on the insights from this work, the opportunity arises to create more meaningful visualisations regarding the dataflow in system performance profiling.
An essential concept of this thesis is to build upon the concepts of tailored profiling to gain a deeper understanding of the system's performance and support the location of potential optimization possibilities. Thus, we integrate an intuitive and interactive query plan visualisation feature that is able to break down complex queries into their constituent operators and pipelines. We clarify further details about the query plan in section 2.3.1 and in chapter X (Implementation) Todo: Chapter linking.

2.3 Related visualisation tools

This section explores related visualisation tools that aid developers analyse their database system queries, with a specific focus on performance visualisation. We will go through the Query Plan Difference Visualiser and the Umbra Profiler Todo: Zitat, which are both tools, that are strongly related to the Benchy Viewer.

2.3.1 Query Plan Difference Visualiser

The efficiency of a database system's query execution relies on the physical execution plan it generates. Given the complexity of finding the best plan, the comparison of query plans both within a single system and accross different systems has garnered attention. This comparative analysis aims to gain valuable insights and identify potential optimisation opportunities. Previous efforts in this direction have mainly focused on quantitative metrics, in particular the total cost of the plan.

Query execution plans describe the step-by-step hierarchical sequence of physical operations that a database system uses to process a particular SQL query. The fascinating aspect of this is that identical queries can result in different plans when processed by different database systems. This variability in plan generation can have a significant impact on overall performance.
The Query Plan Difference Visualiser is a web application that compares and visualises these physical query execution plans from different relational database systems, as shown in Figure 2.1. It is designed for database developers who want to inspect the correlation between variations in query execution speed and the respective query plans. Through enhanced hierarchical differencing algorithms with semantic information about query plans, the tool is able to interactively capture and present the difference between query plans. This is particularly useful for comparing different database systems or different versions of the same system when varying query plans are used

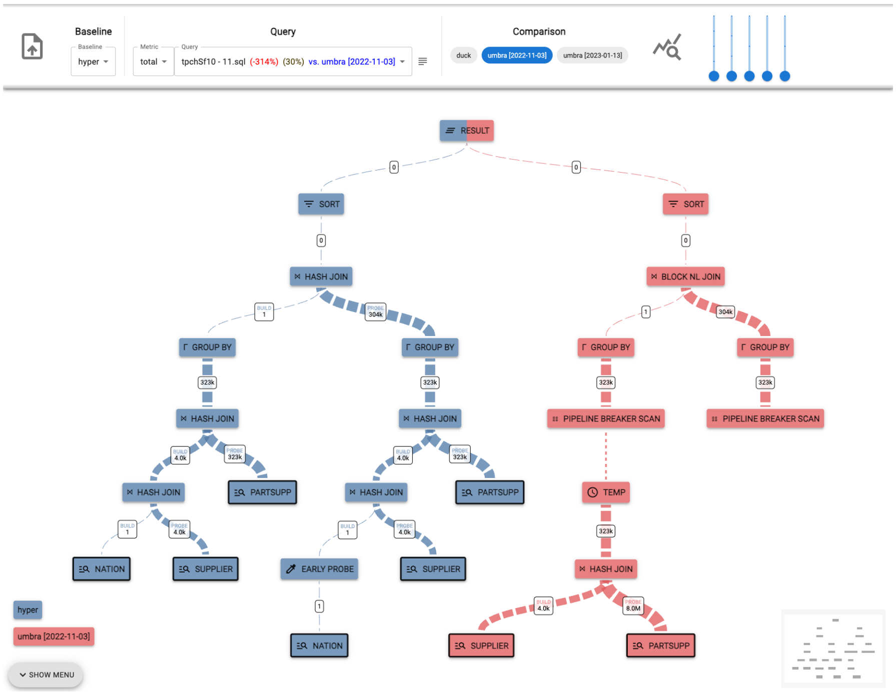


Figure 2.1: Query Plan Difference Visualiser

by the systems under test to process the same query. Furthermore, it provides the flexibility to pick an arbitrary number of systems for which to compare plans and select a metric for evaluating query performance, such as total runtime or compilation time. For the given metric, it directly shows the difference between the baseline system and the better system.

Once a query plan is initialized, the comparative tool provides the option to improve the clarity of the query plan visualisation using various configuration settings. For instance, the tool offers a match mode selection, allowing users to capture the tree from, e.g., a top-down or bottom-up matching perspective. With the Expand and Collapse feature, users can collapse entire subtrees and selectively expand specific child nodes, customizing the focus of the visualisation and tailoring the area of attention to the most interesting parts of the tree. For query plans containing Directed Acyclic Graph (DAG) edges, such as the Pipeline Breaker Scan operator in Umbra, the tool offers support to include these DAG edges and consider them during the matching process. Additionally, to enable more effective comparisons against systems that generate non-DAG-shaped plans, an option is available to replicate subtrees instead.

Database developers often find value in comparing query plans, particularly when they have thoroughly examined different results using quantitative metrics. Therefore, we decided to integrate the Query Plan Difference Visualiser with its core features of comparing query plans. In addition to visualising quantitative metrics from different database systems within the Benchy Viewer, the incorporation of the Query Plan Difference Visualiser with its capabillities will further enhance our objective of simplifying the detection of performance bottlenecks in query execution and optimizing database systems.

Hence, the Query Plan Difference Visualiser is strongly related to this thesis and we will dive deeper into the integration of the comparative tool in Chapter X. Todo: Chapter linking

2.3.2 Umbra Profiler

The Umbra Profiler is a tool that enables in-depth analysis for identifying bottlenecks in query execution processes of the database system Umbra.
It is integrated with a backend application for preparing extensive profiling data and offers multiple perspectives, including a runtime dashboard, a memory dashboard, and an instruction dashboard, each depicting distinct information, as illustrated in Figure 2.2.

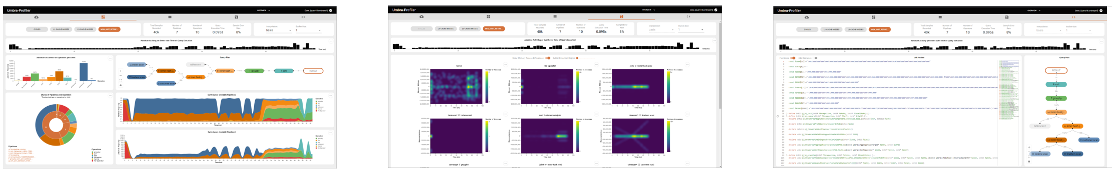


Figure 2.2: Umbra-Profiler: A tool for analyzing and profiling Umbra's compiling queries. From left-to-right: runtime dashboard, memory dashboard, and instruction dashboard.

The runtime dashboard is the default view that appears after providing recorded hardware samples of a query execution. It offers an initial overview of the query execution structure, allowing users to analyze the activity of specific processor events, operators, and pipelines over time. Abnormalities in execution, such as resource-intensive operations, can be identified, aided by visualisations including key performance indicators, activity histograms, bar charts, query plans, sunburst charts, and swim lanes.
The memory behavior dashboard focuses on memory access patterns in query execution. It offers memory heatmaps for each operator, showing either absolute memory accesses or sequential memory address differences.

The instruction Dashboard facilitates detailed analysis of query execution using Umbra Intermediate Representation (UIR) Zitat instructions. It allows comparison of UIR instructions with query plans to identify performance problems based on costs and occurrences.
Similar to the Benchy Viewer, the goal is to support database engineers in optimizing query execution by providing an interactive user interface, enabling an effective in-depth analysis process.
The Umbra Profiler is designed based on the innovative Tailored Profiling approach [Bei+21], where the connection between query plans and compiled code is maintained. This technique was previously unaddressed by standard profilers and is now used in both the Umbra Profiler and the Benchy Viewer.
However, unlike the Benchy Viewer, the Umbra Profiler is focused to operate exclusively with the database system Umbra. In contrast, the Benchy Viewer has the versatility to function with multiple database systems or multiple instances of a single database system.
In broad terms, the Umbra Profiler is primarily designed for in-depth analysis of query performance within a single database system, while the Benchy Viewer is oriented towards its comparative function, enabling the comparison of queries executed by different instances. This comparative approach is the main essence of the concept of the Benchy Viewer, which aims to enhance the understanding of differences between database instances.

An effective scenario that synergizes the Benchy Viewer and the Umbra Profiler would involve identifying intriguing queries using the Benchy Viewer and subsequently conducting comprehensive analyses using the Umbra Profiler.

3 Theoretical Foundations

In this chapter, we investigate the theoretical foundations by examining database performance measurements and in the next step describing used datasets and the data structure.
We will start by discussing the characterisitcs of database systems and elaborate on the significance of performance measurements in this context. Additionally, we will outline the common performance metrics, that play a central role in the evaluation of performance analysis.
For clarifying used datasets and the data structure, we commence by describing the utilized performance data, followed by giving an overview of the structure of the Benchy Viewer's input file containing the performance measurements. Moreover, the data prepration for this input file will be explained.

3.1 Database Systems and Performance Measurements

Performance measurement and analysis are fundamental in the realm of database systems. They offer valuable insights into system behavior, helping to pinpoint bottlenecks and optimization opportunities. This process is crucial not only for evaluating one's own system but also for making meaningful comparisons with other systems. Visualisations play a pivotal role in understanding performance data and are often used to convey complex findings effectively. To interpret performance data effectively, we begin by understanding the characteristics and core traits of a Database System.

3.1.1 Characteristics of Database Systems

Database systems are complex structures that manage and store vast amounts of data efficiently, involving interrelated factors that must be finely tuned to ensure optimal performance.
One of the fundamental functions of a database system is query processing. A query is essentially a request for specific information from the database. This involves receiving and then executing that query. The process includes tasks like parsing, optimization,

and execution.
Queries go through two main phases: compilation and execution. Figure mit Compilation und Execution During compilation, the query is transformed into an execution plan. This plan outlines the steps the system should take to retrieve the requested data. In the execution phase, the system follows this plan to fetch the data.
Query plans are roadmaps that guide how a database executes queries, with operators as specialized components responsible for specific actions. Operators, like selection and join operators, perform data operations during query execution, such as filtering and combining data. Optimizing query plans is vital for database efficiency, with query optimizers selecting the best plan considering factors like data distribution and hardware capabilities.
Query processing can be time-consuming due to various challenges. For instance, complex queries, large datasets, and suboptimal query plans can lead to slow performance. Identifying and overcoming these challenges is essential for improving system efficiency.

Understanding these characteristics is key to understanding the complexity of database performance. Challenges such as optimising query plans and dealing with large data sets are common, and manual assessment is often impractical.
The need for objective metrics is therefore obvious, making performance measurements essential for targeted optimisations. Due to the complexity of these metrics, visualisation techniques are invaluable for easier interpretation and analysis.
In the next section, we will explore the important role of performance measurements and their visualisation in improving database efficiency.

3.1.2 Importance of Performance Measurements

Database systems are the core of a wide range of applications. Consequently, their performance matters not just in terms of user-friendliness and reliability, but also in terms of efficiency. Performance measurements play a central role in this context.
One of the key advantages of performance measurements lies in their capacity to assist in optimization efforts. By quantifying performance in a series of metrics, database developers can pinpoint precisely where bottlenecks occur, whether it is in the compilation phase, the query plan, data retrieval, or any other component of the database system. This focused approach minimizes the trial and error often involved in performance tuning and directs resources toward the most impactful modifications.
Furthermore, bottlenecks and areas with room for improvement are often not obvious. With the aid of performance measurements, these elements come into sharp focus. Measurements can reveal, for example, if the system's weak point lies in query com-

pilation or if the query plan needs to be optimized. Understanding and interpreting the findings correctly is crucial for making informed decisions on where to prioritize improvement efforts.
Another fundamental aspect is scalability. In a world where data is continuously growing, the scalability of a database system becomes a certain priority, because data volumes continue to grow. Performance measurements can identify the limitations of a system as it scales, revealing performance degradation points before they become critical bottlenecks. This approach is not only applied to solve current needs of data volume, but is also contributing to the system's scalability for the future.
Referring back to the introduction's implication, the complexity of performance metrics can often be overwhelming. Visualisation techniques become invaluable tools in this context. By translating numerical data into graphical elements, these visualisations can illuminate patterns and trends that could otherwise be easily overlooked, offering an intuitive and interactive way to understand the performance bottlenecks and operational nuances.

In summary, performance measurements are essential in the effective management and optimization of complex database systems. With these basic principles in mind, the next section examines common performance metrics for evaluating database systems, which serve as the quantitative backbone for the analyses and visualisations discussed here.

3.1.3 Common Performance Metrics

Understanding the importance of performance measurements in database systems necessitates a deeper dive into the specific metrics that help analyse various aspects of performance. This section explores effective metrics and how they are used within this domain, which indicates the desired functionalities of the Benchy Viewer in terms of interaction and visualisation.
In the paper "Bringing Compiling Databases to RISC Architectures" [Gru+23] the compilation performance of the dominant x86-64 server architecture is contrasted with the new introduced code generator designed for AArch64-based systems. This is interesting for the Benchy Viewer as it conducts a comparative analysis of different perspectives in terms of performance, leveraging specific performance metrics that are also visually represented.
The paper utilizes both quantitative and subjective performance metrics when addressing the query compilation strategy. However, for the scope of our visualisation, we focus on the quantitative metrics. Relying on quantitative metrics allows for clear, objective visualisation that can represent performance differences, whereas subjective metrics

does not offer the same level of clarity and consistency in a visual representation.
Here one of the most central metrics is the throughput, a key metric in databases, measures the number of processed tuples per second and is a primary optimization target. In the context of compiling databases, throughput is primarily influenced by the quality of the generated machine code for queries.
Another fundamental metric is the latency, which is the time needed for generating and compiling query code before execution, with lower latency being particularly important for real-time transactional systems.
With these two metrics, the paper shows an intuitive and clear overview of how different database instances perform on the TPC-H benchmark, as demonstrated in Figure 3.1.

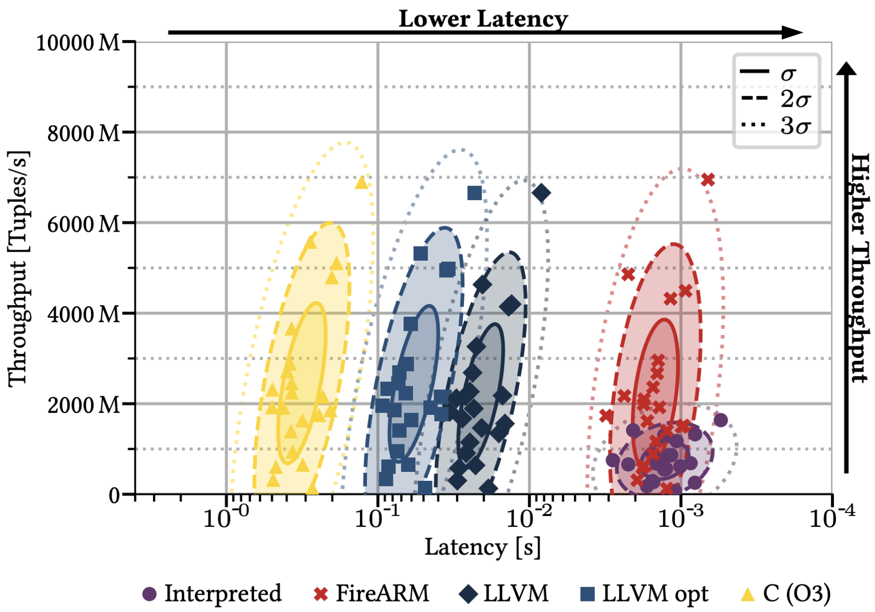


Figure 3.1: Visualisation example of compile-time and throughput of different query-compilation strategies running the TPC-H benchmark [Gru+23].

The visualisation presents a scatter plot that groups query results into clusters, with each cluster representing a database instance by a distinct color. The Y-axis displays the throughput in tuples per second, while the X-axis shows the latency in seconds, which is a descending value from left to right. Additionally, arrowed labels point to the preferred values: higher values on the Y-axis and lower on the X-axis. Thus, top-right-corner values represent optimal performance, allowing viewers to quickly identify well-performing database instances as well as the performance differences

between instances.
In this illustration, the system highlighted in red is notable for its low latency and high throughput. Conversely, the system marked in yellow has the poorest latency performance. The instance in purple also stands out, boasting the lowest latency, but it lags behind in terms of throughput.

In the context of performance metrics, the Benchy Viewer should be capable of visualising the key differences between instances in an intuitive and effective way.

Besides grouping the query results into clusters, with each cluster representing a database instance, it would also be beneficial to have the flexibility to choose a specific metric for this categorization. For instance, if a database developer is primarily concerned with the metrics of execution time and throughput, they should have the option to shape the data visualisation based on these metrics.

Up next, we'll explore the dataset that is consumed by the Benchy Viewer to offer all the data and performance metrics within the analytical visualisations. We'll detail the data structure and how the data is prepared to get in this shape.

3.2 Used Datasets and Data Structure

In this section, we talk about the dataset that is consumed by the Benchy Viewer to create meaningful performance visualisations.
At first, we explore the diverse metrics encompassed within the dataset. For each metric, we provide a concise definition and rationalize its significance within the Benchy Viewer framework. Subsequently, we offer an outline of the input file's structure, where we explain how database systems are mapped to the executed queries and the corresponding metric data. Finally, we describe all the steps required within the data preparation process, where raw data is transformed into the previously specified format.

3.2.1 Description of the Utilized Performance Data

In this section, we take a closer look at the performance data which is utilized by the Benchy Viewer. These data contain various metrics that give us insights into how our database system is performing: total time (compilation, execution), cycles, instructions, L1 data cache misses, LLC (Last-Level Cache) misses, branch misses, DTLB (Data Translation Lookaside Buffer) misses, tasks, instructions per cycle (IPC), CPU frequency (GHz), and scalability metrics.

Total time represents the combined time taken for both query compilation and query execution. It's a critical measure of how efficiently queries are processed. Lower total times are desired, indicating faster query processing.

Compilation is the preparation for the execution phase where source code, written in a high-level language, is converted into a lower-level representation. Producing highly-optimized code in this stage can significantly enhance the speed and efficiency of program execution.

Execution refers to the phase after the compilation phase where the compiled code is executed on a CPU. It involves the actual processing of instructions and data to perform the tasks specified by the program. The execution phase is of vital significance as it directly measures how efficiently a program or task runs on the hardware.

Instructions count the number of individual machine-level instructions during the execution of a program or a specific operation and helps assessing the efficiency of code execution. A high instruction count indicates that the program is performing a large number of computations, which impacts CPU utilization and overall performance. Well-optimized code tends to have a lower instruction count for the same computational tasks. While instruction count provides valuable information about code complexity, it does not capture the complete performance picture. For a more comprehensive understanding, it should be considered alongside metrics such as cycle counts and IPC (Instructions Per Cycle).

Cycles refer to the number of clock cycles executed during the test by a CPU. It measures the raw computational effort involved and can indicate the CPU's workload. Lower cycle counts indicate more efficient code execution, while higher counts suggest greater computational complexity or inefficiencies. While cycles provide valuable information about computational effort, they do not give a complete picture of overall system performance. Other metrics, such as instructions per cycle (IPC), may be necessary to better understand the performance landscape.

IPC (Instructions Per Cycle) is a performance metric that measures the average number of instructions during a single clock cycle. A higher IPC value indicates that the CPU is executing more instructions per clock cycle, which suggests better performance. Well-optimized code and algorithms tend to have higher IPC values.

L1D-Misses (Level 1 Data Cache Misses) are a performance metric that counts the number of times a CPU requested data from its Level 1 Data Cache but was unable to find the required data there. Instead, the CPU had to retrieve the data from a higher-level cache or main memory. The number of L1D-Misses is significant because it reflects how efficiently the CPU's cache hierarchy is operating. High L1D-Miss rates suggest that the CPU frequently needs to access data from slower memory levels, resulting in increased latency and potentially impacting overall system performance. Lower L1D-Miss rates generally indicate more efficient cache utilization and can result in improved execution performance.

LLC-Misses (Last-Level Cache Misses) are a performance metric that counts the number of times a CPU failed to find requested data in its last-level cache. Instead, it had to retrieve the data from a slower memory hierarchy level, such as a higher-level cache or main memory. The number of LLC-Misses is significant because it indicates how effectively the CPU's last-level cache is utilized. Similar to L1D-Misses high LLC-Miss rates suggest that frequently accessed data is not readily available in the cache, leading to increased memory access latency and potential performance bottlenecks. Lower LLC-Miss rates generally indicate more efficient cache utilization and can lead to better overall execution performance.

Branch-Misses, often referred to as "branch mispredictions," are a performance metric that counts the number of times a CPU incorrectly predicts the outcome of a branch instruction. Branch instructions are conditional statements (e.g., if-else or loops) in code that determine the program's flow based on a condition. A branch miss occurs when the CPU's branch predictor guesses incorrectly about the branch's outcome and, as a result, has to discard or re-execute some instructions. Therefore, high Branch-Miss rates can indicate inefficiencies in the code execution, as mispredicted branches can lead to the execution of unnecessary instructions and decreased overall performance. Several factors influence Branch-Miss rates, including the complexity of code logic, the CPU's branch prediction algorithms, and the effectiveness of compiler optimizations. Other metrics, such as instruction count, cycle count, and cache utilization, should be also considered to obtain a comprehensive view of performance.

DTLB-Misses (Data Translation Lookaside Buffer Misses) are the number of times a CPU requested data from memory, and during this request, it also needed to fetch or translate the virtual memory address to its corresponding physical memory address, but couldn't find the translation in the Data Translation Lookaside Buffer (DTLB). Instead, it had to consult a more extensive translation structure, such as the page table in memory, to perform the translation. The number of DTLB-Misses is significant because it indicates how effectively the CPU's DTLB, which is responsible for accelerating memory address translation, is functioning. High DTLB-Miss rates suggest that the translation process is less efficient, leading to increased memory access latency.

Tasks refer to concurrent units of work or threads that a computer system or application is managing or executing simultaneously. These tasks may represent various processes, threads, or parallel workloads. The number of tasks and their management is significant because it reflects the system's concurrency and workload handling capacity. They help assess how well a system scales with increased workloads and concurrent tasks.

CPUs are defined by the number of cores utilized by the system. A higher number of CPUs implies a higher degree of concurrent computing, potentially leading to executing the given query in a more efficient manner. However, the effectiveness of parallel processing also depends on factors like architecture, clock speed, and the specific nature of the computing tasks.

GHz (Gigahertz) is a unit of measurement used to quantify the clock speed or frequency of a CPU or other electronic components within a computer system. Specifically, it represents one billion cycles per second. In computing, it is primarily used to describe the operating frequency of a CPU. The GHz metric is significant because it reflects how quickly a CPU can process instructions and perform calculations. Higher GHz values generally indicate faster processing speeds, which can lead to quicker execution of tasks and improved system performance.

Scale, in the context of performance analysis for databases, refers to the size of the dataset. It quantifies the system's ability to handle increasing amounts of data and workload. Scale is often measured in terms of data volume, storage capacity, or concurrent user connections. Scale-related metrics are essential for benchmarking and comparing different database systems, hardware configurations, or software designs. They help assess how systems perform as data or workload sizes increase and enable informed decisions regarding system scalability.

In this section, we have provided a comprehensive overview of the performance metrics that will be employed in our system. The next section will detail how we format and organize our performance data for analysis.

3.2.2 Structure of the Input File with Performance Measurements

One step preceding the utilization of the Benchy Viewer is the provision of the benchmark data, which will be later used to generate the performance visualisations. In this section, we give an overview about the structure of the input file for the Benchy Viewer.

When submitting data to the Benchy Viewer, only one CSV file is needed to be uploaded to the system. This CSV file contains the benchmark data of all the participating database systems.

The structure of the input file, as depicted in Figure 3.2, is well-defined. It contains the entries of the respective database systems, with each query being associated with a database system. For example, when comparing TPC-H benchmark data, each database

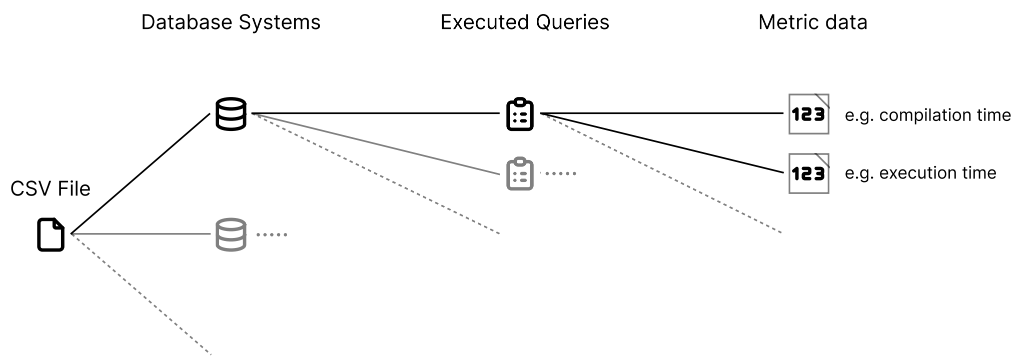


Figure 3.2: CSV structure of the input data for the Benchy Viewer

should comprise data for all 22 TPC-H queries. Each individual query contains data for multiple metrics, which were defined in the previous section such as compilation time, execution time, cycles, instructions, etc.

Moreover, the system accommodates the use of multiple instances of a single database system. Within a database system entry, there is an attribute that allows the specification of the system's particular version. Besides, comparing different systems, this feature enables the comparison of different configurations within one system.

The queries contain metric data which are expressed in specific units or data types. Time-related metrics, such as total, compilation, and execution times, are measured in milliseconds, offering insights into the temporal aspects of query processing.
Hardware-related metrics, including cycles, instructions, cache misses, and more, are provided as integer values, reflecting various hardware-level details. Task-related metrics are also presented as integers, helping to assess task-specific performance. IPC metrics are in floating-point numbers, offering a nuanced perspective on instruction efficiency. CPU-related metrics are integers, while frequency-related metrics are in gigahertz (GHz), providing information about CPU clock speeds.

Next up is the "Data Preparation" section, where we'll dive into the necessary steps for formatting and structuring our data to make it compatible with the Benchy Viewer.

4 Implementation

Chapter Introduction

4.1 Concept: Features and Interaction Capabilities of the App

In this section, we explore the expected features and capabilities crucial for our objectives and the necessary interaction capabilities from an end user's perspective.
We start with the visualisation capabilities of the benchmark data, where the goal is to easily identify performance bottlenecks. This involves pinpointing interesting queries with distinct performances for subsequent in-depth analysis.
For deeper analysing and achieving a profound understanding of diverse performance results within queries, the application should support a focus on individual queries. Users should be able to view the performance of a single query from various perspectives, providing a comprehensive image. Comparative analysis across queries from different database systems or even the same database but with different configurations is also essential.
Then, we examine the requirements for a suitable dashboard that contains the main part of the performance visualisations. Multiple views are necessary to construct a complete understanding of the current point of interest. The optimal structure of these views may vary depending on the point of interest. The flexibility of a drag-and-drop solution for structuring views allows users to tailor the overview according to their specific interests.
Finally, we examine the capability of the system to facilitate collaboration and knowledge sharing. The application should enable the saving and sharing of potential findings. This involves saving the interface configuration of the analysis for sharing with other users. Recipients should be able to upload the received configuration, gaining access to the same data and visualisation structure for collaborative exploration.

4.1.1 Visualising Benchmark Data

Visualising Benchmark Data includes multiple aspects. It involves the import of the actual data, where a proper way of importing the data should be provided. We also cover the utilized plots and charts for the data visualisation. In addition to visualisation, we offer a comprehensive table view of the data.

Charts and Plots

We aim to offer a dashboard containing multiple data views. To prevent UI clutter, a universal legend of database system instances is suitable, eliminating the necessity to display a legend in every chart. In the legend depicted in Figure 4.1 all database systems included in the benchmark data are presented as coloured toggles, allowing users to activate or deactivate their appearance in the visualisation. We dive deeper into the interactive legend in 4.1.2.

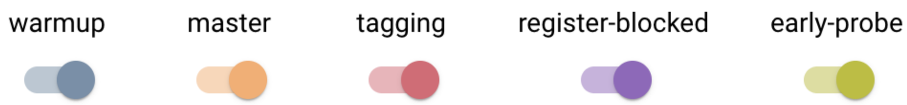


Figure 4.1: Gobal legend of all database systems contained in the imported performance data.

When initiating the analytical process with the Benchy Viewer after importing the benchmark data, it may be beneficial to start with a general and straightforward overview of the benchmark data for quickly gaining an understanding of the general performance standings of the database systems.
Within the Benchy Viewer, this overview is effectively conveyed through the relative performance visualisation, as depicted in Figure 4.2. This chart displays, on the Y-Axis, the relative performance of each system in percentage compared to the best-performing system in the benchmark data. The X-Axis lists and represents every database system contained in the benchmark data.

After obtaining this initial overview, users can quickly identify the system marked with the red colour as the benchmark, as it processes all queries faster than all other systems. In this visualisation, this system is set as the benchmark with 100% performance, providing a clear reference point for relative comparisons.
This chart serves as a valuable starting point for further in-depth analyses and allows users to grasp the overall performance landscape before delving into specific metrics or

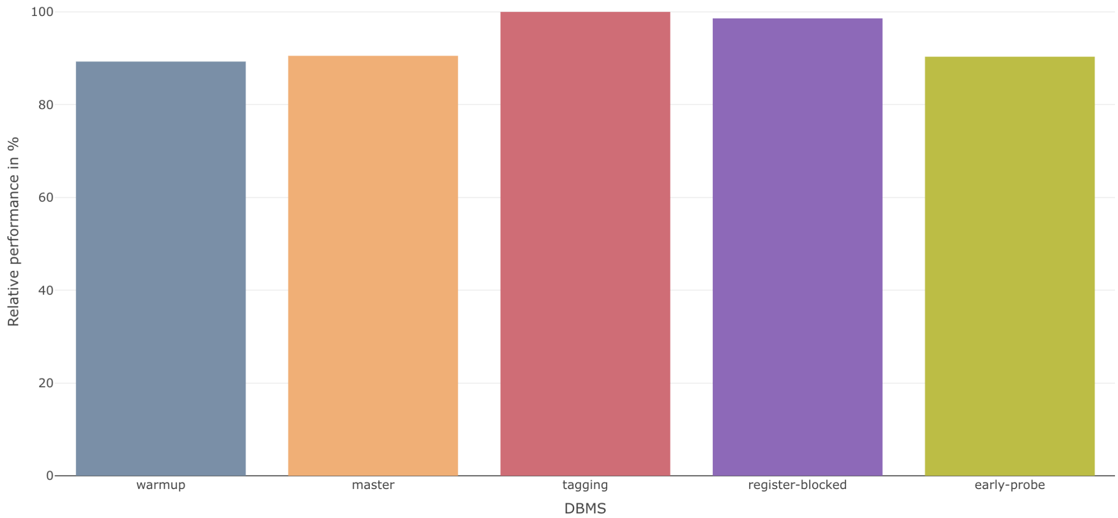


Figure 4.2: Bar chart visualises the relative performance of all systems compared to the best performing system.

queries.

In the next phase of the analysis process, we delve into two crucial aspects: compilation and execution. Compilation is the phase where source code is translated into machine code, preparing it for execution Compilation. It is a crucial step that impacts the overall performance of the system. On the other hand, execution is the phase where the compiled code is run on the system. This step involves the actual processing of the instructions and the generation of results. The time spent in the execution phase is a key factor in determining the system's efficiency in handling tasks.

To understand how much time each database system allocates to these processes, the Benchy Viewer offers a stacked bar chart, illustrated in Figure 4.3. Here, one section of the bar signifies the time spent in the compilation step, while the other represents the time spent in the execution step.

Using solely the ratio of the two different steps, without taking into account the overall performance, would present an incomplete picture of the balance between the two process steps compared to the better-performing systems. Therefore, this chart complements the relative performance visualization discussed earlier. By incorporating information about the overall performance, it aims to offer a comprehensive understanding while illustrating the equilibrium between compilation and execution for each system.

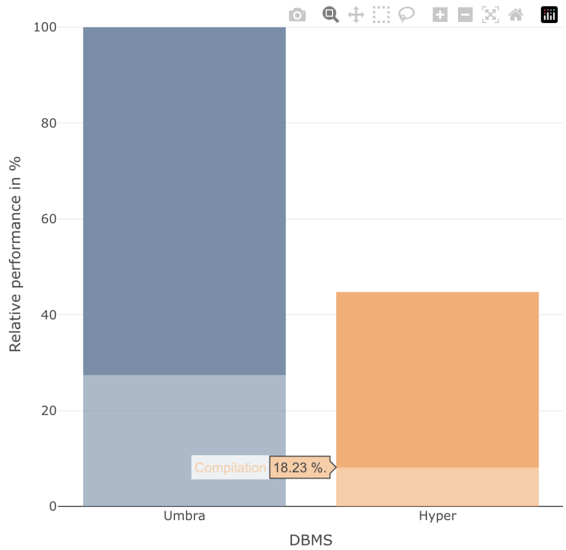


Figure 4.3: Stacked bar chart illustrating the distribution of time between compilation and execution steps. The compilation step is depicted in a transparent accent colour, while the execution step is represented in the full colour intensity.

In this example, a comparison is made between two database systems, with the Y-Axis indicating the relative performance and the X-Axis listing different database systems. Umbra stands out as the best-performing system, with its bar reaching the 100% level. In contrast, Hyper attains 45% of Umbra's performance. When hovering over one of the process steps, the chart displays the percentage of the total time spent by the system in that specific step. For instance, Hyper allocated 18.23% of the entire process to the compilation step, while Umbra spent 27% on this phase.

Hence, the stacked bar chart in the Benchy Viewer offers a consolidated view of how each database system allocates time between compilation and execution steps. Its integration of the relative performance metric ensures a balanced understanding of system efficiency, preventing oversights from focusing solely on step ratios. With clear visual distinctions and the ability to compare multiple systems, this chart enhances analytical insights into performance disparities and offering an aerial view of compilation and execution.

The preceding visualisation solution Referenz auf PDF shows essential benchmark visualisations, offering both a comprehensive overview of all queries and a detailed examination of each individual query.

It commences with a comparative analysis of each query using a bar chart. Similar to

Figure 4.4, this chart displays queries from various database systems on the X-Axis, with the total time presented on the Y-Axis.

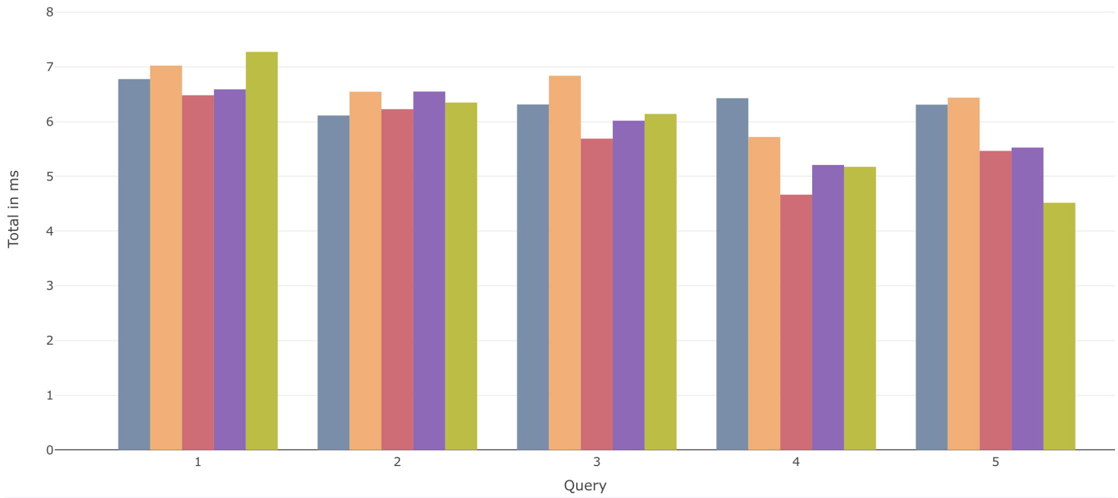


Figure 4.4: Bar chart visualises the totals time in ms of different queries.

Bar plots stand out as a versatile visualisation method, particularly when tasked with presenting the performance metrics of multiple queries. Their inherent clarity, with the length of each bar directly corresponding to a specific performance metric such as compilation time or execution time, makes them well-suited for diverse scenarios.
The straightforward visual comparison they offer is a notable strength. With each query distinctly represented by a separate bar, variations in performance become immediately apparent.
This quality proves especially valuable for our objective of identifying outliers, as these exceptional values are easily noticeable.

To facilitate a quick and clear overview of all queries, violin charts are employed. These charts not only provide an initial glimpse of the overall system performance but also offer distribution insights, including the shape and density of the data. They can also be combined with box plots to get a concise summary of the distribution of the results, displaying key statistics such as the median and quartiles.

Within the Benchy Viewer violin plots that contain data points, as shown in Figure 4.5 on the right side, additionally allow you to hover over a data point. This action highlights the corresponding query in the violins of the other database systems. We will explore this hover feature further in 4.1.1.

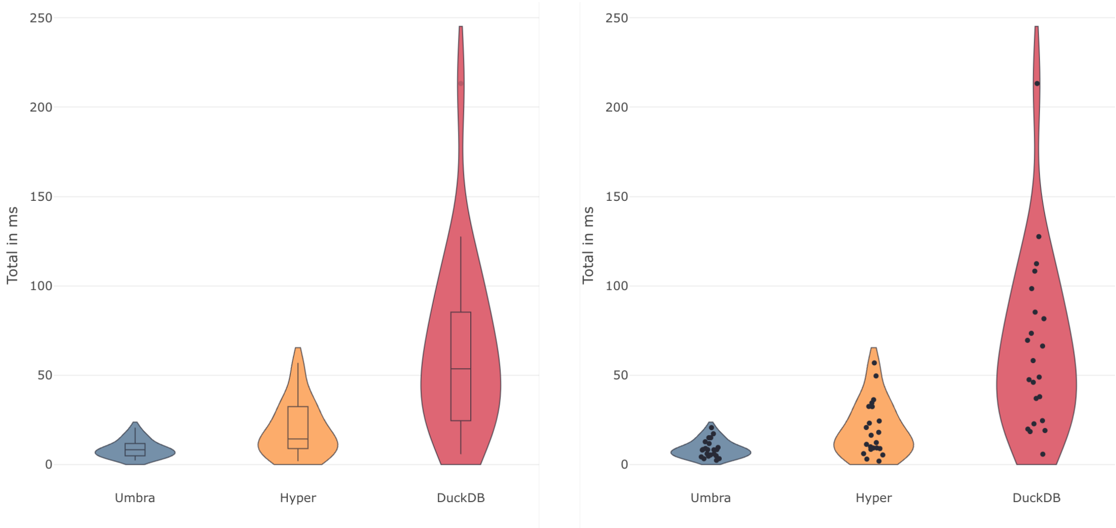


Figure 4.5: Violin charts visualise the totals time in ms of different queries. The left variant contains a box plot and the right variant contains all data points.

Conducting benchmark performance analysis often necessitates a comparative approach between a chosen system and other competing systems. The Benchy Viewer facilitates this by enabling the selection of a baseline system from one of the database systems included in the benchmark data, as shown in Figure 4.6.

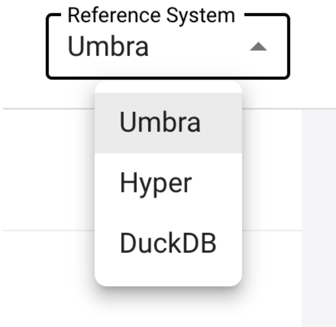


Figure 4.6: A drop-down menu that allows to select a baseline system.

This functionality proves useful for assessing how the performance of the chosen baseline system compares to others, aiding in the identification of strengths, weaknesses, and potential optimization areas. It forms a foundational step in the detailed analysis provided by the Benchy Viewer. Several visualisations in the application utilize the baseline system, including the table view and scatter plot for inspecting the slow-down and speedup metric, and the bar chart showcasing the performance delta for each query from the perspective of the baseline system. All these visualisations depend on a baseline system, making this functionality crucial for a comprehensive performance analysis.

In the context of providing a clear and comparative understanding of how different systems perform relative to a chosen baseline, the metrics maximum slow-down and maximum speedup become crucial. They offer a comprehensive view of the range of performance variations. Maximum slow-down indicates the worst-case scenario of reduced performance, while maximum speedup highlights the most significant improvement achieved.

Slowdown indicates how much slower a specific system is compared to the baseline system. It is calculated as the ratio of the time taken by the system under consideration to the time taken by the baseline system. A slow-down value greater than 1 implies that the system is slower than the baseline. For example, a slow-down of 1.5 means the system is 1.5 times slower than the baseline.
Identifying slow-downs is crucial for pinpointing areas of inefficiency or performance bottlenecks in a system. It helps in understanding where improvements are needed.

Speedup, on the other hand, quantifies how much faster a specific system is compared to the baseline system. It is calculated similarly to slow-down but in the reverse manner. A speedup value greater than 1 implies that the system is faster than the baseline. For example, a speedup of 2 means the system is twice as fast as the baseline.
Knowing the speedup is essential to highlight improvements. It indicates the effectiveness of optimizations or enhancements made to the system compared to the baseline.

The Benchy Viewer employs tables to showcase the maximum slow-down and maximum speedup, visualised in Figure 4.7, where the maximum slow-down is displayed on the left side and the maximum speedup on the right side using Umbra as the baseline system. The use of cell colours in the table serves as an effective visual indicator of performance outliers. Intensity in colour corresponds to the extent of the outlier, offering a rapid understanding of the range and distribution between these extreme values. This colour-coded representation aids users in identifying and assessing the significance of performance variations across different systems.

The tables are organized based on the resulting ratio.
In the maximum slow-down tables, the arrangement is ascending, placing the slowest queries of the baseline system compared to the faster alternative system at the top.
Conversely, in maximum speedup tables, the sorting is descending, presenting the

Maximum slow-down for Umbra

Query	Umbra	Hyper	DuckDB
19	15.03 ms **1x**	6.11 ms **0.41x**	48.95 ms **3.26x**
6	2.37 ms **1x**	1.93 ms **0.81x**	5.79 ms **2.44x**
14	3.28 ms **1x**	3.08 ms **0.94x**	19.85 ms **6.05x**
1	8.44 ms **1x**	18.04 ms **2.14x**	46.05 ms **5.46x**
2	5.37 ms **1x**	34.51 ms **6.42x**	37.98 ms **7.07x**

Maximum speed-up for Umbra

Query	Umbra	Hyper	DuckDB
4	3.32 ms **1x**	9.30 ms **2.80x**	85.31 ms **25.70x**
20	4.69 ms **1x**	9.95 ms **2.12x**	108.26 ms **23.10x**
21	12.77 ms **1x**	20.75 ms **1.63x**	213.26 ms **16.70x**
10	8.35 ms **1x**	32.52 ms **3.89x**	112.42 ms **13.46x**
22	6.01 ms **1x**	8.52 ms **1.42x**	73.46 ms **12.23x**

Figure 4.7: Tables showcase the maximum slow-down and the maximum speedup using colour intensity to indicate performance outliers.

fastest queries of the baseline system compared to the alternative system at the forefront. This sorting strategy provides a logical structure to quickly identify and compare performance differences in either scenario.

The scatter plot is another effective choice for visualising speedup and slow-down. These plots are particularly beneficial for trend analysis, offering a detailed view of each query individually. Users can identify trends, clusters, or outliers in the data, providing insights that might be less apparent in a table.

In the Benchy Viewer a baseline is displayed, as illustrated in Figure 4.8 positioned at Y = 1 and coloured in blue. This positioning allows users to observe the relative performance of each query compared to the baseline system.

In this example, the scatter plot represents the maximum speedup for the selected baseline system on the Y-axis, while the total time in milliseconds is represented on the X-axis. This visualises the relationship between the total time taken by each query and its corresponding speedup compared to the baseline system.
Hovering above these query data points reveals the ratio of the corresponding speedup and the query identifier. We will explore the hover feature further in 4.1.1.
Queries from the competing system are marked in orange, and most of them are slower, with three queries below the baseline, indicating that these particular queries are faster. This observation prompts further investigation into the queries below the baseline, offering an opportunity to identify potential performance bottlenecks from the perspective of the baseline system. This nuanced understanding, facilitated by the visual representation of the scatter plot, guides users in pinpointing specific queries that deviate from the expected performance trend, aiding in targeted optimizations and analysis.

In the Benchy Viewer, another visualisation occurs, highlighting the performance

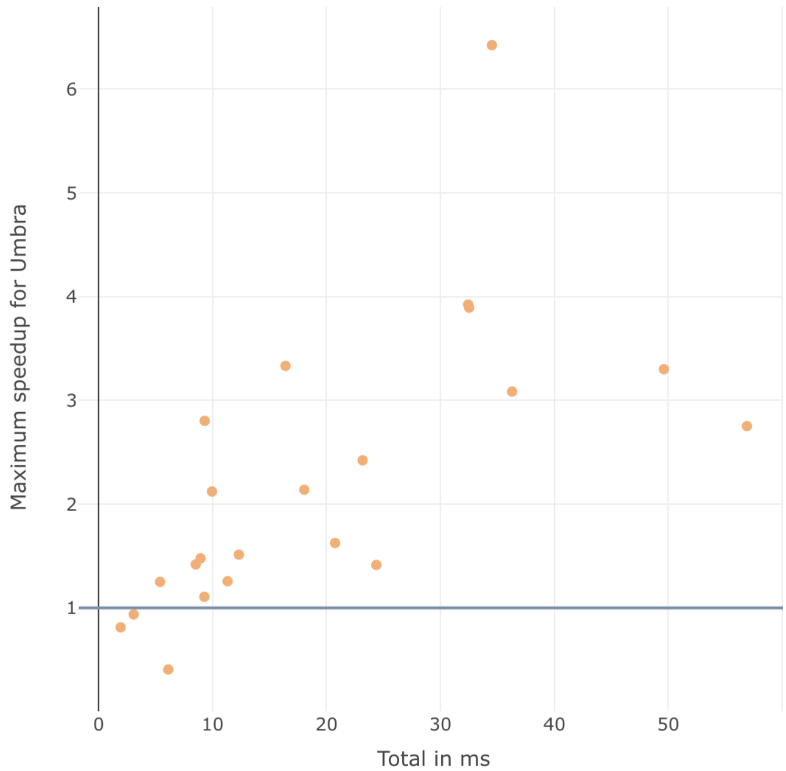


Figure 4.8: Scatter Plot visualises the speedup.

disparities between a chosen baseline system and its competitors. It takes the form of a bar chart that illustrates the performance gap for each query, providing a perspective of the baseline system compared to the best-performing query of other competing systems.

The performance delta chart in the Benchy Viewer not only displays the performance gap for each query from the perspective of the baseline system but also indicates the best competing system for each query by using corresponding system colours, as depicted in Figure 4.9. The Y-axis shows the performance gap percentage of the competing queries, while the X-axis represents the identifying query number. Hovering over bars reveals the exact performance delta.

In this example, the red-marked database system is selected as the baseline system. Queries with faster performance by the baseline system are displayed above the baseline, indicating a positive performance delta. In contrast, queries with stronger performance by competing systems are shown below the baseline, indicating a negative performance

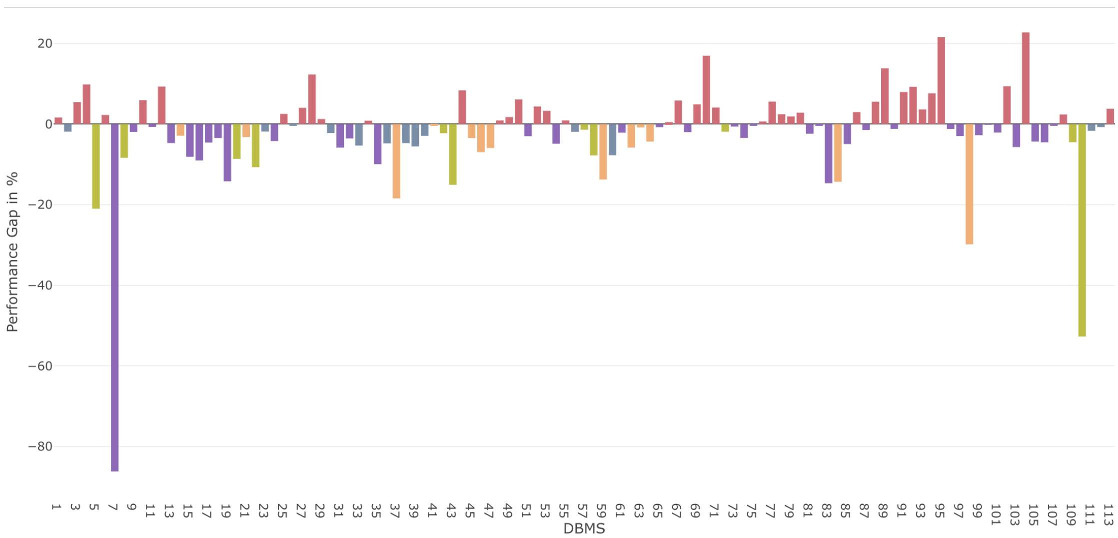


Figure 4.9: Bar chart visualises the performance gap for every query of the baseline system compared to best corresponding query of the competing systems.

delta for the baseline system. Additionally, the bar of each query is marked with the colour of the corresponding best-performing database system.
The performance gap for the majority of cases in this example stays within a range between -20% and 20%. However, some outliers are notably significant. For instance, the seventh query has a performance gap of -86%. A deeper analysis of this query may be sensible to potentially identify performance bottlenecks from the perspective of the baseline system.

This query performance gap visualisation is instrumental in quickly discerning how each query of the baseline system stacks up against the top-performing queries of other systems. The chart's simplicity ensures easy comprehension, enabling users to promptly assess the relative performance of different queries.

Hover Feature

Hovering over data points in a chart is an indispensable feature, commonly employed to extract more detailed information about a specific data point. In the context of analysing query performances and identifying noteworthy queries within visualisations, the ability to scrutinize a particular query from multiple perspectives becomes crucial.

In the Benchy Viewer, we've elevated this feature, which is illustrated in Figure 4.10, to

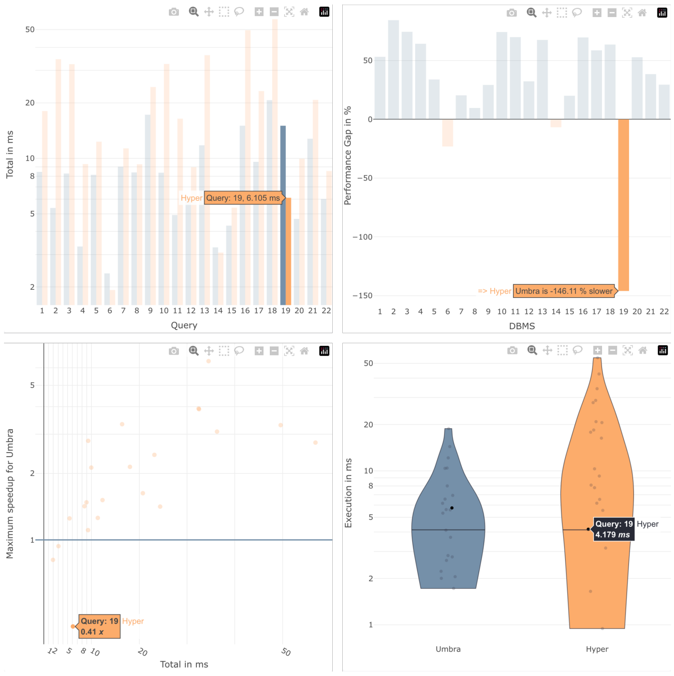


Figure 4.10: Hovering over a query automatically highlights in a global context the same query in all visualisations which are showcasing single queries. These visualisations include violin plots, scatter plots, and bar charts when one bar represents a single query.

a global hover capability within the application.
This means that when a user hovers over a specific query, not only is that query highlighted within the current chart, but the same queries within all other visualizations are also highlighted simultaneously.This synchronization spans various visualizations, including violin plots, scatter plots, and bar charts which represent distinct queries.

Data Viewer

In addition to visualisations, the Benchy Viewer features a data table for in-depth inspection of benchmark data. This table presents the imported user data, as exemplified in the excerpt of the data viewer in Figure 4.11.

This view displays the complete data, which is described deeply in section 3.2.2.
The header contains all properties and metrics of a data row. The data rows below the header represent the data of a query, containing the reference to the database system

title	dbms	version	query	total	total_mean	total_median
Umbra	umbra	HEAD	1.sql	[8.79572000000…	8.437	8.401
Umbra	umbra	HEAD	2.sql	[5.28696, 5.2973…	5.372	5.319
Umbra	umbra	HEAD	3.sql	[8.20781, 8.3119…	8.264	8.244
Umbra	umbra	HEAD	4.sql	[3.42394, 3.3200…	3.319	3.279
Umbra	umbra	HEAD	5.sql	[8.13269, 8.0803…	8.13	8.107
Umbra	umbra	HEAD	6.sql	[2.367946, 2.371…	2.371	2.368

Figure 4.11: Snapshot of the Data Viewer: Organized rows and columns of benchmark data.

and the complete benchmark data of this query.
The user can scroll vertically and horizontally given the amount of the data rows and the metrics. The header provides some options in a drop-down menu to work with the data sheet, as shown in Figure 4.12. The first option is to sort the table by a chosen data column containing numerical data. The user has the possibility to do this ascending or descending and also reset the sorting. This is useful when the user wants to sort specific data, e.g. the total time in ms.

This view showcases the benchmark data, elaborately described in Section 3.2.2. The header encompasses all properties and metrics of a data row. Data rows below the header represent the data of a query, including the reference to the database system and the complete benchmark data for that query.
Vertical and horizontal scrolling are enabled to navigate through the extensive data rows and metrics, as the volume of information exceeds the screen's capacity.

The header offers various options in a drop-down menu to interact with the data sheet, as depicted in Figure 4.12.

The first option enables sorting the table based on a selected data column with numerical values. Users can choose between ascending or descending order, and there's an option to reset the sorting. This functionality is particularly useful when organizing specific data, for instance, sorting the table based on total time in milliseconds.

The filter option, as shown in Figure 4.13a, provides a tool for refining the displayed data in the table. Users can easily narrow down their focus by selecting a specific column, an operator, and a value. This allows for targeted exploration of the data, such as isolating rows related to a particular database management system, as demonstrated

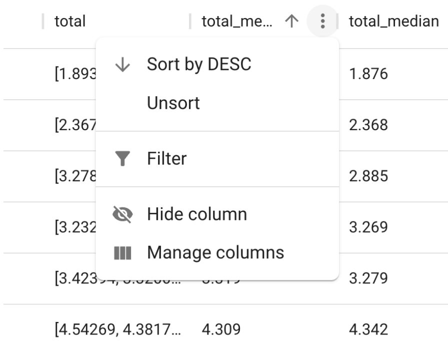
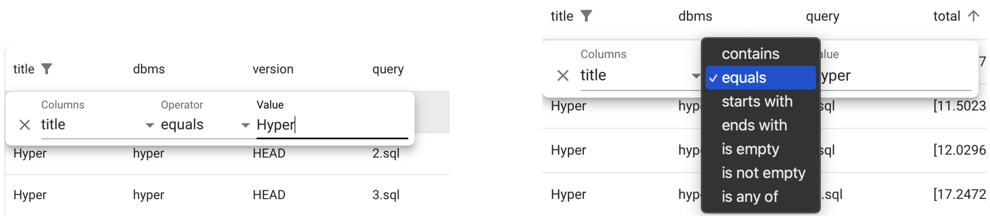


Figure 4.12: Column options drop-down offering sorting, filtering, and column visibility functionality.



(a) Filter Menu.

(b) The list of available operators.

Figure 4.13: Filter menu allows data sheet filtering by column, operator, and value.

in Figure 4.13b. This filtering capability enhances the user's ability to extract meaningful insights from the benchmark data.

When dealing with extensive datasets, the Benchy Viewer ensures flexibility and ease of use by allowing users to tailor their view. Through the 'Manage columns' feature, accessed via the drop-down menu, users can not only hide columns but also gain a holistic overview of active columns.

Figure 4.14 showcases this functionality, providing users with the ability to effortlessly activate or deactivate columns, conduct quick searches, and streamline their view by hiding or showing all columns at once. This empowers users to focus on the specific data points relevant to their analysis, enhancing the efficiency of the benchmark data exploration process.

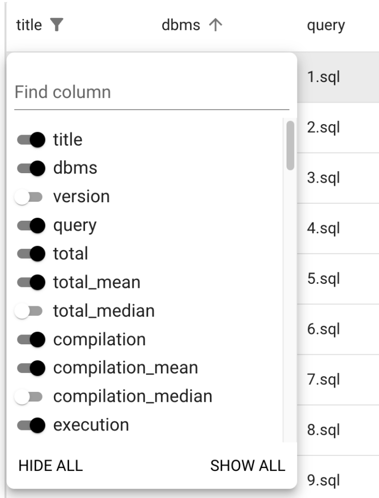


Figure 4.14: Column Manager: Easily control column visibility in the data sheet for a tailored view.

4.1.2 Query Analytics: Comparative Examination and Comprehensive Benchmark Insight

One of the primary functions of the Benchy Viewer is to conduct a comparative analysis of the performance across various database systems. The application offers a range of tools for this purpose. In addition to presenting diverse charts and plots for analyzing query data from various angles, the Benchy Viewer allows users to inspect the query plan of a selected query in a comparative manner. To initiate this process, users need to choose the particular query and the involved database systems.

Preparation for In-Depth Analysis

In the analytical process of identifying and inspecting a noteworthy query, it is beneficial to refine the information for a more precise and clear understanding.
The Benchy Viewer features an interactive legend that not only provides information about group colors but also offers the ability to enable or disable all query visualizations of a system throughout the entire application.

This is particularly valuable when a significant query is identified, and a comparison between two systems is of interest, while there are other less relevant systems in the data that may clutter the visualizations. In such cases, users can utilize the interactive header, illustrated in Figure 4.15, to streamline their view and prioritize the visualizations

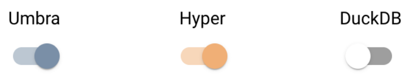


Figure 4.15: Interactive legend with the functionality to activate or deactivate database systems.

based on current needs.

In the course of analysing benchmark data and identifying noteworthy queries, a deeper inspection of these queries becomes essential. The Benchy Viewer incorporates a versatile hover feature, as introduced in Section 4.1.1, providing a multi-perspective view of a specific query by highlighting it across various visualizations simultaneously.

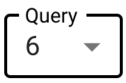


Figure 4.16: Drop-down for selecting a specific query for deeper inspection.

Similar to this functionality, the Benchy Viewer provides the option to directly select a specific query. This can be accomplished through various means, such as using a drop-down menu, as illustrated in Figure 4.16, or by clicking on a specific query data point within a visualisation (e.g., clicking on a bar in a bar chart or a data point in a scatter plot).

Selecting a query will also result in highlighting the query in all visualisations, similar to the universal hover feature, also depicted in the Figure 4.10. So, it is possible to highlight two distinct queries at once, the selected query and the hovered query, which enables a comparison between two queries using multiple perspectives.

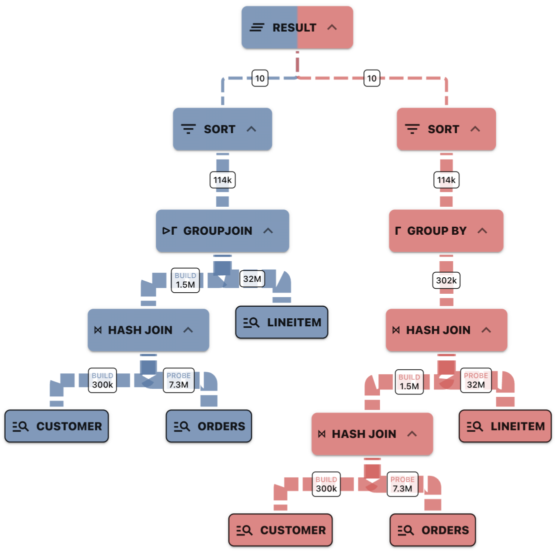
Choosing a specific query results in its highlighting across all visualisations, similar to the universal hover feature demonstrated in Figure 4.10. This enables users to concurrently emphasize two separate queries, the chosen query and a hovered one, enabling a comparative analysis from various angles.

Comprehensive Query Insights

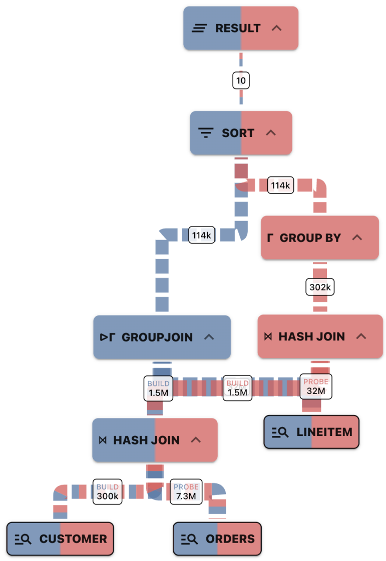
In the analysis workflow facilitated by the Benchy Viewer, once a significant query has been identified for more in-depth examination, the subsequent step involves selecting this particular query and transitioning to the Query Plan View. In this phase, users gain the ability to meticulously inspect the query plan associated with the chosen query for each database system under consideration.

The interactive legend, depicted in Figure 4.15, maintains its presence in this view, allowing users to seamlessly activate or deactivate the display of query plans corresponding to specific database systems.

For clarity and comparative analysis, all activated query plans are rendered in a unified visualization. Each query plan is presented in its designated system colour. This representation is exemplified in Figure 4.17a, where the various query plans are harmoniously displayed for comprehensive comparative scrutiny. This visual approach not only streamlines the analysis process but also contributes to a more intuitive understanding of the comparative performance of queries across different database systems.



(a) Query plan view comparing the query plan of a specific query between two database systems.



(b) Query plan view comparing the query plan of a specific query between two database systems.

Figure 4.17: Query plan comparison between different database systems.

The query plan feature is equipped with many tools designed to facilitate a thorough inspection of query plans. A central capability is the merging of all activated query plans using a specified strategy, using the slider illustrated in Figure 4.18. This merging process, exemplified in Figure 4.17b, involves summarizing identical subtrees across different plans. Such merging proves invaluable for inspecting divergences in inspected queries, as varying query plan structures are more clearly visualized. For an in-depth exploration of merging strategies, please refer to Section 4.5.

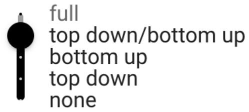


Figure 4.18: Selection of the query plan merging strategy in the form of a slider.

For extensive query plans, it's beneficial to conceal subtrees that are presently less relevant. This action reduces the overall size of the query plan, emphasizing the more pertinent sections. Users can employ this functionality for any node, as illustrated in Figure 4.19.

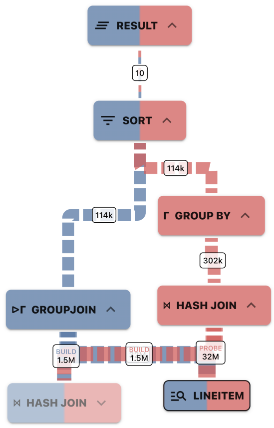


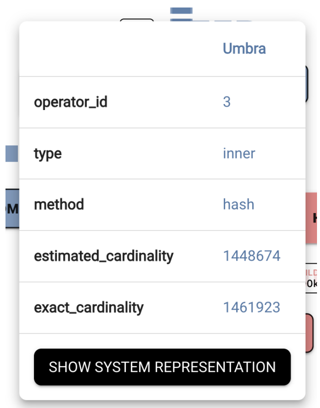
Figure 4.19: Query plan with a hidden subtree in the bottom left corner.

When dealing with extensive query plans, having a tool to comprehend the entire canvas is essential. To facilitate this, a map, as shown in Figure 4.20, is included in the bottom right corner of the query plan view. This map provides an overview of the query plans using simplified visual representations of the nodes. Furthermore, it indicates the current location within the complete layout of the query plan, aiding in orientation.

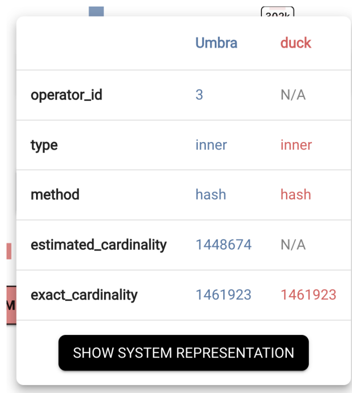
The query plan offers supplementary details about nodes. Clicking on a node opens a concise overview, as depicted in Figure 4.21a, displaying information about the operator,

Figure 4.20: Map of the query plan showing the current location and the tree in a simplified version.

its type, estimated cardinality, and exact cardinality.
In scenarios where multiple databases are activated, and a merged node is selected, the information view shows details for all systems being merged within that node, as illustrated in Figure 4.21b.



(a) Node information window.



(b) Node information window of a shared node.

Figure 4.21: Node information window of the selected node.

The information window also allows displaying the system representation. As an overlay, similar to Figure 4.22, the system representation appears in the style of a simple code editor, presenting information in a code-like format. Line numbers are displayed on the left, and an overview of the file is shown on the right.

At the top, the user can choose the database system for which the system representation should be displayed in the case of a merged node.

All these functionalities are provided by the semantic diff tool ZITAT. We dive deeper into the integration of the semantic diff tool in the section 4.5.

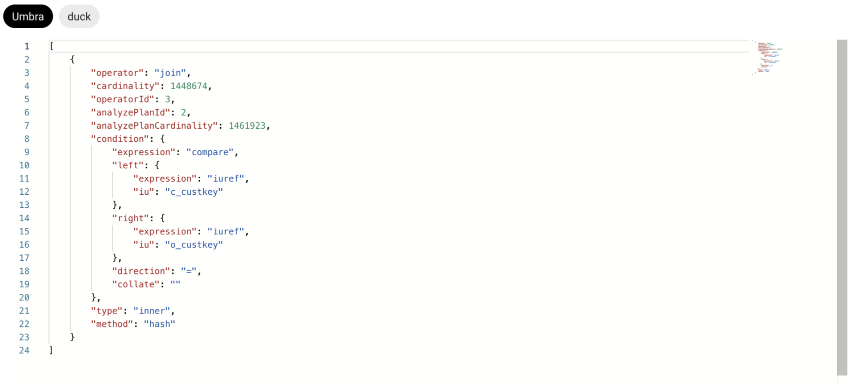


Figure 4.22: Query plan system representaion of a node.

4.1.3 Flexible Interface Hub

To offer a well-suited platform for analysing specific performance differences of high complexity, a flexible system is essential for conducting these analytical processes. Such flexibility is crucial for providing tailored solutions to inspect specific aspects of benchmark data.

In this section, we will explore how the Benchy Viewer achieves this functionality, primarily through a drag-and-drop system for visualization elements. This empowers users to select the necessary visualisations and construct a comprehensive overview.

Subsequently, we will delve deeper into the actual visualisations, examining the flexibility of configuring charts and plots to optimize the perspective of performance data within a specific visualization.

Drag and Drop

The drag-and-drop system integrated into the Benchy Viewer empowers users to effortlessly rearrange every chart and plot. Beyond the flexibility offered by individual visualization elements, users can strategically organize visualizations within containers. These containers are essentially thematic sections, each represented by a headline. Users have the flexibility to rename these containers, creating dedicated spaces for specific topics and populating them with relevant charts and plots.

The versatility of containers allows them to function as distinct analytical sections, housing visualization elements pertinent to that specific context. Moreover, the dynamic nature of the Benchy Viewer enables users to seamlessly move visualizations between

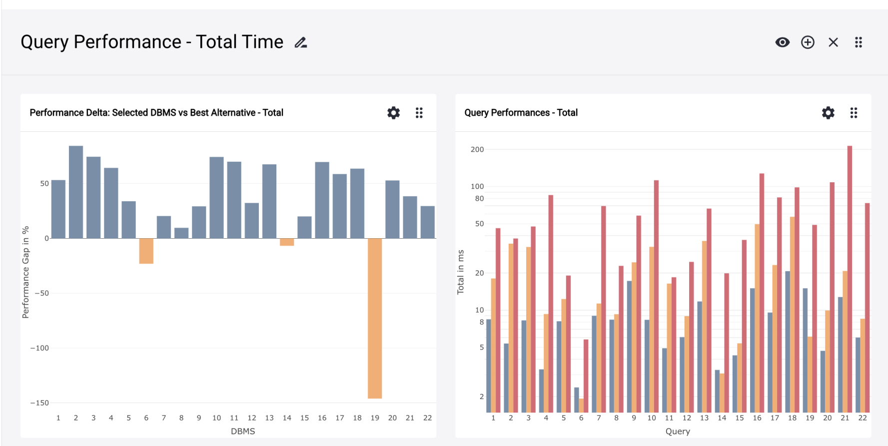


Figure 4.23: Flexible interface hub through drag and drop feature for visualisation elements and their containers.

containers, fostering adaptability in organizing and categorizing data. The ability to drag entire containers further enhances this adaptability, enabling users to arrange sections according to their analytical preferences. This dual flexibility, within containers and with container placement, provides users with a robust framework for tailoring their analytical environment to meet the demands of diverse datasets.

Analytics Sections

The top header of a container, as illustrated in Figure 4.23, serves as a hub for fundamental container functionalities. An intuitive feature allows users to rename a container by clicking on the edit icon situated next to the container's label. This action triggers a dialogue with a straightforward text input, as depicted in Figure 4.24. Once users input and confirm the new label using the "ok" button, the container's label promptly updates.
Users can now efficiently organize and contextualize their analytical elements within appropriately named containers. The ability to swiftly modify container labels enhances the user experience, enabling them to maintain a well-structured and easily navigable analytical workspace.

To enrich the content within a container, users can effortlessly incorporate charts and plots using the icon positioned in the right corner of the container's header. This action opens an overlay menu presenting a variety of available plots and charts for inclusion in the container, as showcased in Figure 4.25. Upon the user's selection of a visualisation

Figure 4.24: Text input for renaming containers.

element, the analytical section promptly updates, seamlessly integrating the chosen element into its content.

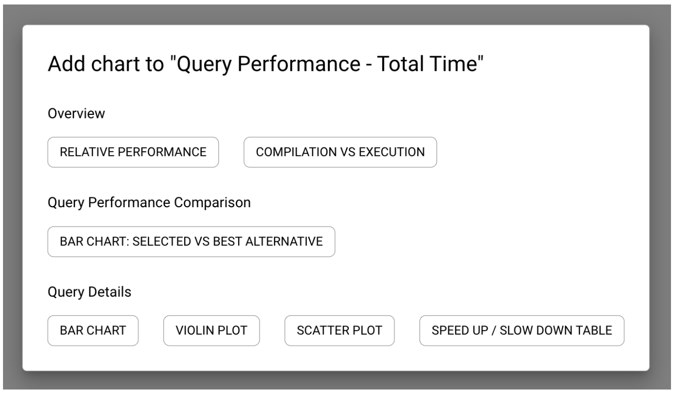


Figure 4.25: Overlay for adding charts to the current container.

This intuitive process streamlines the augmentation of analytical sections, allowing users to dynamically enhance their workspace with relevant visualisations. The added charts and plots are instantly ready for utilization, fostering a fluid and responsive analytical environment within the Benchy Viewer.
We dive deeper into the utilization and configuration of the charts and plots in 4.1.3.

Another valuable functionality in the container's header is the visibility toggle. Users can swiftly conceal an analytics section by simply clicking the visibility icon of a container, as illustrated in Figure 4.26. This action disables the view of the content, revealing only the header of the container. Users can easily reactivate visibility or access other functionalities.

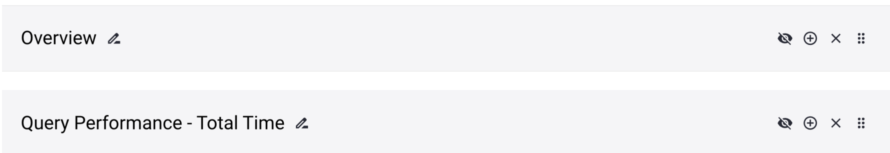


Figure 4.26: Visibility property to hide or show a container.

The convenience of collapsing containers, coupled with the drag-and-drop feature, enhances overall user comfort and efficiency. This uncomplicated feature significantly enhances clarity and ease of use, especially when managing a substantial number of visualizations and diverse data perspectives.

Initiating the deletion process, another noteworthy feature embedded in the container's header is the delete function. By clicking on the delete icon of a container, users can efficiently remove an analytics section. This action permanently eliminates the container and its content. This straightforward delete function provides a quick and effective way to manage the organizational structure of visualizations.

Chart Configurations

Within the Benchy Viewer, a versatile dashboard serves as a backdrop for seamless interaction with diverse visualisations. Delving into individual visualisation elements, the platform prioritizes adaptability, allowing users to tailor charts and plots to their specific requirements.
We'll explore the capability to select a metric within a chart, demonstrate the flexibility of switching to a table visualisation in the context of maximum speed-up and maximum slow-down, and finally, highlight global chart options.

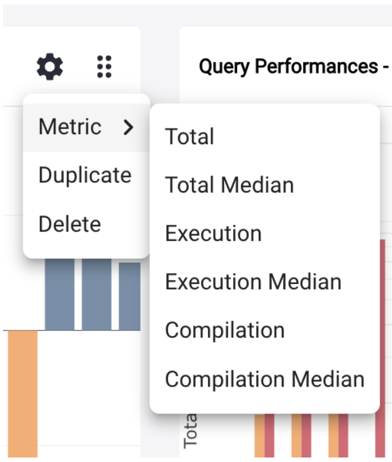


Figure 4.27: Drop-down menu of chart options offering to select the visualised metric. Also the delete and duplicate functionality is provided.

Every visualisation element within the drag-and-drop dashboard features a header, as illustrated in Figure 4.23. Beyond the drag-and-drop functionality, the header hosts an options icon, providing access to various configuration settings, as depicted in

Figure 4.27.
Within the nested drop-down menu, users can seamlessly tailor their selection of metrics to meet specific needs.

Upon selecting a metric, the chart dynamically updates, presenting the new data within the current visualisation framework. This rapid and flexible construction of charts enhances the user's ability to gain valuable insights swiftly.

Similarly, users can quickly duplicate a chart for comparative analysis or delete it to streamline the dashboard. The duplicate function allows for the replication of a chart with its current settings, providing an efficient way to explore variations of the same data.
Conversely, the delete function promptly removes a chart, contributing to a clutter-free and focused visual workspace. These functionalities collectively empower users to fine-tune their analytical environment for a seamless exploration experience.

In the analytical process of working with the metric pair speed-up and slow-down, seamlessly toggling between maximum speed-up and maximum slow-down is crucial. The Benchy Viewer facilitates this transition through the switch icon integrated into the header of the table visualization, as exemplified in Figure 4.28.

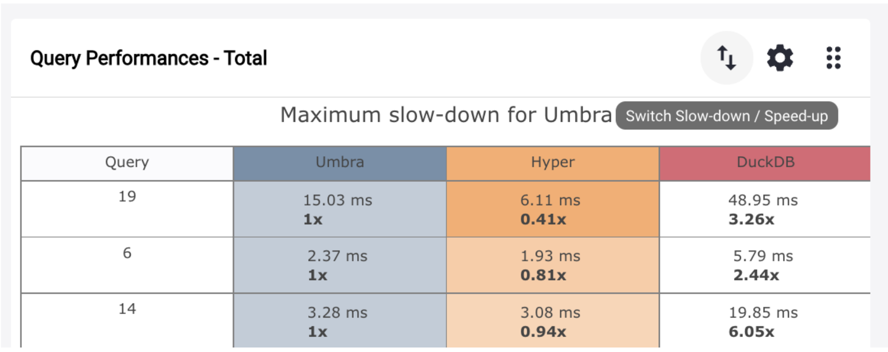


Figure 4.28: Table visualisation provides the functionality to switch between maximum speed-up and maximum slow-down.

A simple click on the switch icon triggers the table to transition to the alternate mode, dynamically updating its data to reflect the chosen metric. This feature ensures a fluid and efficient exploration of performance data, allowing users to effortlessly switch between speed-up and slow-down perspectives as needed.

In addition to the nuanced control over individual visualizations, the Benchy Viewer empowers users with global chart options, allowing users to sculpt the visual landscape to their specific analytical requirements.

One of the noteworthy features is the ability to globally switch the Y-axis type, as depicted in Figure 4.29. This flexibility allows users to choose between linear, logarithmic, and throughput representations, offering diverse perspectives on the benchmark data. Whether aiming for a detailed examination of variations in lower values or emphasizing proportional changes, the global Y-axis type switch ensures that the visualizations align with the user's analytical focus.

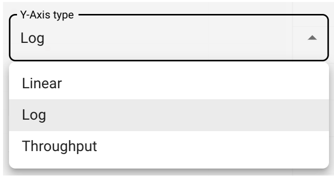
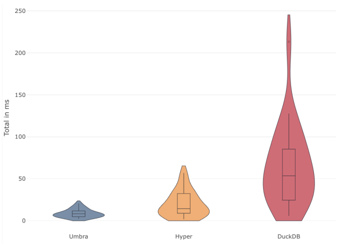
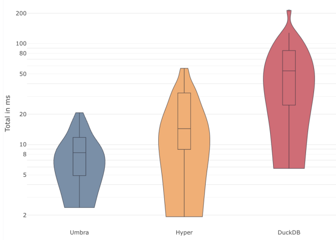


Figure 4.29: Drop-down menu of global options offering the selection of the scale type.

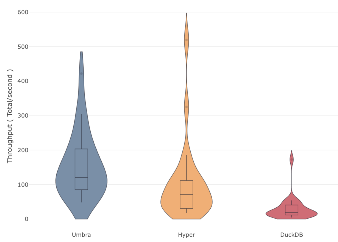
The linear scale is ideal when a precise examination of small variations in data is paramount. This scale excels in offering a detailed, granular view, particularly advantageous for closely positioned metric values.



(a) Violin plot with linear scale.



(b) Violin plot with logarithmic scale.



(c) Violin plot with throughput based scale.

Figure 4.30: Comparison of violin plots using different scales: A visual representation of the same dataset, highlighting how variations appear under linear, logarithmic, and throughput scaling.

The logarithmic scale excels in highlighting proportional changes across a wide range of values. This is particularly advantageous when dealing with datasets that span several orders of magnitude, ensuring that both small and large values are perceptible.

The throughput scale is specifically designed for scenarios where the emphasis is on the rate of data transfer or processing. It provides a unique perspective, crucial in benchmarking scenarios where throughput is a critical performance metric.

By seamlessly switching between these Y-axis types globally, users can extract diverse insights from the same set of data, enhancing the versatility and depth of their analytical processes.

The Benchy Viewer also offers customization options for violin plots. Users can choose between representing data inside the violins as individual data points or as a boxplot, as illustrated in Figure 4.31. Opting for data points means that each query data is individually represented, providing a wealth of detailed information. On the other hand, selecting the "Boxplot" option replaces individual data points with a summarized boxplot inside the violin. This option streamlines the visualization, offering clarity by presenting an overview of the data distribution. Users have the flexibility to tailor the representation based on their specific analytical needs.

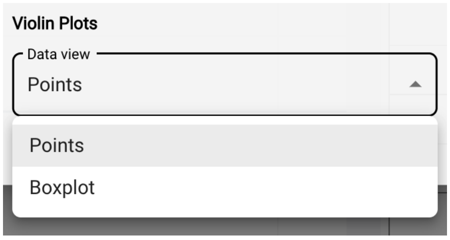


Figure 4.31: Drop-down menu of global options, offering the selection of displaying data points or a boxplot within the violins of the violin charts.

4.1.4 Saving and Sharing the Application State

4.2 Design

4.2.1 User Interface

4.2.2 Page Structure and Navigation

4.3 Data Structure

4.3.1 Overall Project Structure

4.3.2 Input File and Benchmark Data

Import of Performance Data

The first step, working with the Benchy Viewer is to import a file containing the performance data which should be visualised. The file needs to follow the format introduced in Section 3.2.2. Figure 4.32 summarizes the import of the input file in a process diagram.

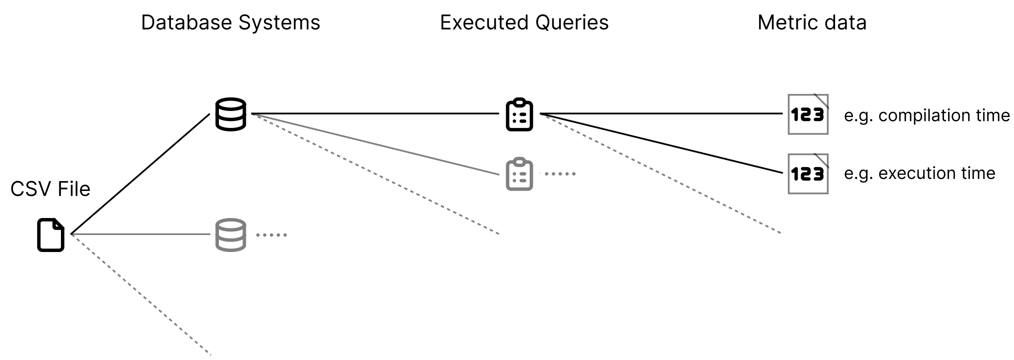


Figure 4.32: Import process of input data

Chart beschreiben.

4.3.3 Plot Options

4.3.4 Visualisation Arrangement Data Structure

4.3.5 Query Plan

Visualisation Parameters

Query Plan Data Structure

4.4 Integration of Plotly-React for Data Visualisation

4.4.1 Types of Plots and Charts

4.4.2 Hover Feature

4.4.3 Selected Query Feature

4.5 Integration of semantic-diff-tool

4.5.1 Business Logic

4.5.2 Settings

4.5.3 UI

5 Discussion

5.1 Evaluation of Achievement of Objectives

5.2 Critical Reflection on the App Development

5.2.1 Challenges/ Technical Limitations (Performance limits)

5.2.2 Design Choices and Trade-offs

5.2.3 User Feedback and Iterative Development

5.2.4 Comparison with Existing Solutions

5.2.5 Potentials and Future Improvements

6 Conclusion

6.1 Summary of Results

6.2 Future Developments and Enhancements

6.3 Final Remarks (Summary of Key Findings/ Contribution to the field)

7 Example

7.1 Section

Citation test [Lam94].

Acronyms must be added in `main.tex` and are referenced using macros. The first occurrence is automatically replaced with the long version of the acronym, while all subsequent usages use the abbreviation.

E.g. `\ac{TUM}, \ac{TUM}` $\Rightarrow$ Technical University of Munich (TUM), TUM

For more details, see the documentation of the `acronym` package[1].

7.1.1 Subsection

See Table 7.1, Figure 7.1, Figure 7.2, Figure 7.3.

Table 7.1: An example for a simple table.

A	B	C	D
1	2	1	2
2	3	2	3

[1] `https://ctan.org/pkg/acronym`

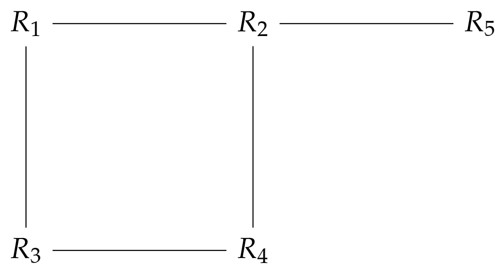


Figure 7.1: An example for a simple drawing.

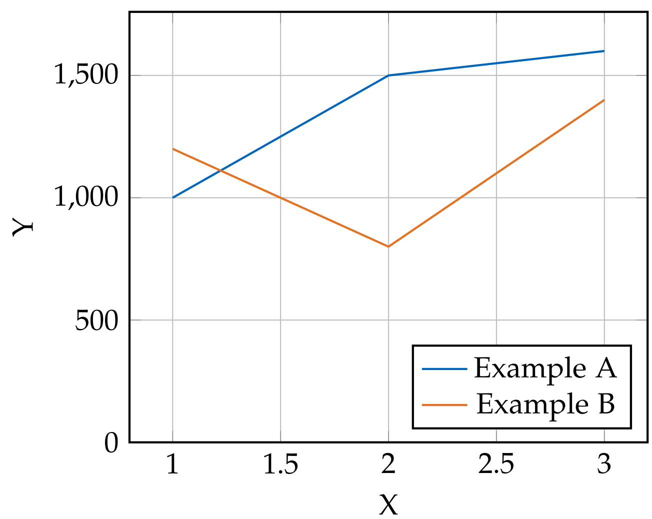


Figure 7.2: An example for a simple plot.

```
SELECT * FROM tbl WHERE tbl.str = "str"
```

Figure 7.3: An example for a source code listing.

Abbreviations

TUM Technical University of Munich

List of Figures

List of Tables

Bibliography

[Bei+21] A. Beischl, T. Kersten, M. Bandle, J. Giceva, and T. Neumann. "Profiling dataflow systems on multiple abstraction levels." In: Apr. 2021, pp. 474–489. DOI: 10.1145/3447786.3456254.

[Gru+23] F. Gruber, M. Bandle, A. Engelke, T. Neumann, and J. Giceva. "Bringing Compiling Databases to RISC Architectures." In: *Proc. VLDB Endow.* 16.6 (Apr. 2023), pp. 1222–1234. ISSN: 2150-8097. DOI: 10.14778/3583140.3583142.

[Lam94] L. Lamport. *LaTeX : A Documentation Preparation System User's Guide and Reference Manual*. Addison-Wesley Professional, 1994.