SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Visualization 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

Visualization 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:	15.12.2023

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

Munich, 15.12.2023 Julian Macias De La Rosa

Acknowledgments

Abstract

Contents

1 Introduction

Interactive performance visualization 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

1.3 Existing Visualization 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

The optimization of queries is a pivotal aspect in the realm of database performance profiling, and various existing approaches are dedicated to supporting this field with visualization tools. This chapter offers an overview about the existing work in the domain of the visualization of database performance profiling. We will investigate the importance of optimizing query executions in database systems and the role of visualizations 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 visualization tool closely associated with this thesis, as its key feature is integrated into our tool, the Benchy Viewer.

2.1 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 visualizations regarding the dataflow in system performance profiling.
An essential concept of this thesis is to develop and provide effective performance visualizations built upon the concepts of tailored profiling, which should contribute to a

better understanding of the system's performance and support the location of potential optimization possibilities. Thus, we integrate an intuitive and interactive visualization tool that is able to break down complex queries into their constituent operators and pipelines. We clarify further details about the implementation in Chapter 4.

2.2 Related visualization tools

This section explores related visualization tools that aid developers analyse their database system queries, with a specific focus on performance visualization. We will go through the Query Plan Difference Visualiser [23k] and the Umbra Profiler [Bei+21], which are both tools, that are strongly related to the Benchy Viewer.

2.2.1 Query Plan Difference Visualiser

The efficiency of a database system's query execution relies on the 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.

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 [FN21].
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 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.

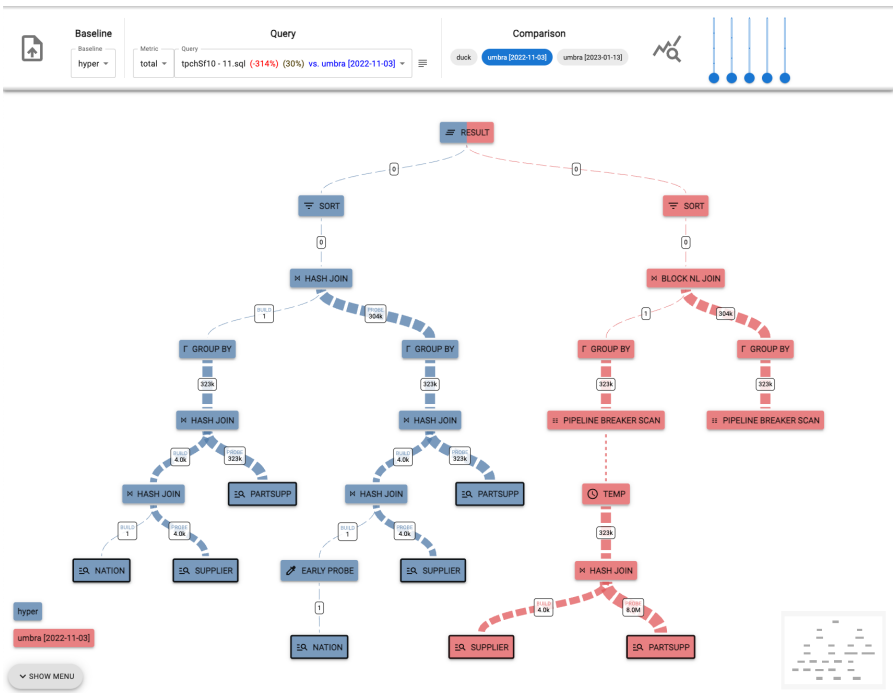


Figure 2.1: Query Plan Difference Visualiser.

Once a query plan is initialized, the comparative tool provides the option to improve the clarity of the query plan visualization 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 visualization 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 with other systems, particularly when they have thoroughly examined different results using quantitative metrics and cardinality estimates. 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 capabilli-

ties 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 4.5.

2.2.2 Umbra Profiler

The Umbra Profiler 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.

(a) Runtime Dashboard. (b) Memory Dashboard. (c) Instruction Dashboard.

Figure 2.2: Umbra-Profiler: A tool for analyzing and profiling Umbra's compiling queries.

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 visualizations 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) [KLN21] 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. 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 comprehensive performance assessment not only allows for the effective evaluation and refinement of one's own system but also provides a benchmark for meaningful comparisons with other systems, fostering a continuous cycle of improvement. Visualizations 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, intricate in design, efficiently manage large datasets, ensuring fast data retrieval and maintaining consistency, which guarantees reliability of stored information. Optimization through meticulous tuning is crucial for overall system

effectiveness. 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.
In a compiling database management system, queries go through two main phases: compilation and execution [Neu11]. 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 recipes 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, visualization techniques are invaluable for easier interpretation and analysis.
In the next section, we will explore the important role of performance measurements and their visualization 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 consistency 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 compilation 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.

Referring back to the introduction's implication, the complexity of performance metrics can often be overwhelming. Visualization techniques become invaluable tools in this context. By translating numerical data into graphical elements, these visualizations 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 visualizations 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 visualization.
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 visualization, we focus on the quantitative metrics. Relying on quantitative metrics allows for clear, objective visualization 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. It quantifies the number of processed tuples per second and stands as a primary target for optimization. In the context of compiling databases, throughput is primarily influenced by the quality of the generated plan and machine code.
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.

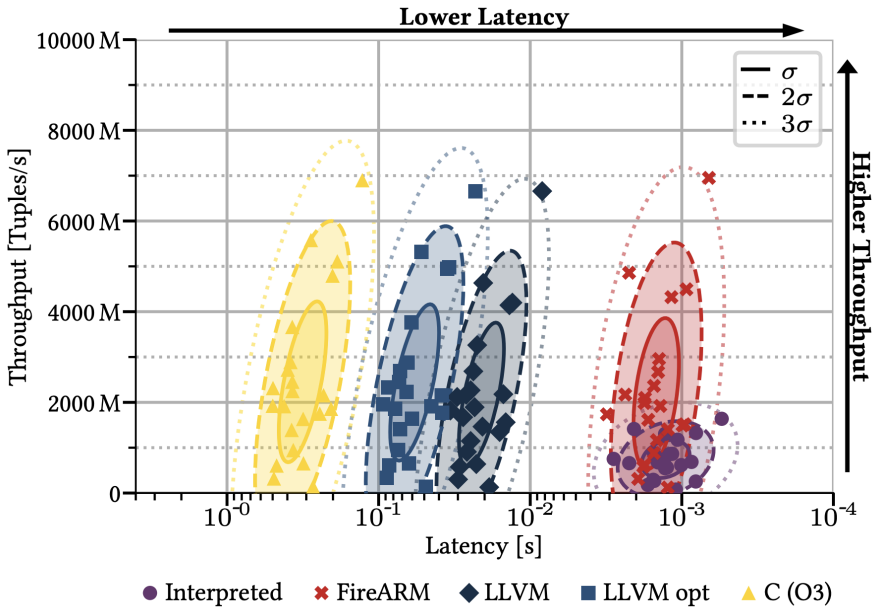


Figure 3.1: Visualization example of compile-time and throughput of different query-compilation strategies running the TPC-H benchmark [Gru+23].

The visualization presents a scatter plot depicting data points that represent compilation times in the context of throughput and latency. Clusters are formed by grouping data points, with each cluster representing a distinct backend and distinguished by color. The Y-axis illustrates throughput in tuples per second, while the X-axis indicates latency in seconds, descending from left to right. Arrowed labels highlight preferred values, guiding attention to higher values on the Y-axis and lower values on the X-axis.
Taking into account the inverted scale of the X-Axis, where latency is portrayed, the top-right corner becomes the indicator of optimal performance. This allows viewers to

swiftly identify backends that perform well and discern performance disparities among instances.

In this illustration, the FireArm in red is notable for its low latency and high throughput. Conversely, the C backend has the poorest latency performance. The LLVM backend 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.

In addition to organizing query results into clusters, where each cluster corresponds to a specific database instance, providing the flexibility to select a specific metric for this categorization would be advantageous. For example, if a database developer's primary focus is on throughput concerning the compilation and execution, coupled with maximum speed-up, they should be able to tailor the data visualization to reflect these specific 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 visualizations. 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 visualizations.
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 a high-efficiency approach for query execution, utilizing just-in-time (JIT) compilation to create custom machine code for each query [Gru+23; FMT20]. This involves translating the query from a relational plan to an intermediate representation (IR) and then to native machine code. While boosting execution efficiency, it adds a step and incurs translation costs, particularly impacting short or complex queries, leading to longer response times. For smaller datasets, the relative cost increases as most time is spent in compilation.

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.

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.

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.

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 higher-level caches 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 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 base operating frequency of a CPU. However, it's important to note that modern CPUs often have dynamic clock adjustments, such as Intel Turbo Boost [23b]. This feature allows the CPU to temporarily increase its clock speed based on system demands.

Scale is a metric that pertains to the quantity of tuples processed by a database system within a defined pipeline. Tuples denote individual data entries, and the scale metric evaluates the system's effectiveness in handling and manipulating data across different volumes. It serves as a crucial indicator of a database's performance and scalability, with a higher scale value indicating a larger dataset. Typically expressed as the number of tuples processed per unit of time, this metric aids in evaluating throughput and efficiency under diverse workloads, providing insights for optimizing and configuring the system.

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 visualizations. 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 should comprise data for all 22 TPC-H queries. Each individual query contains data

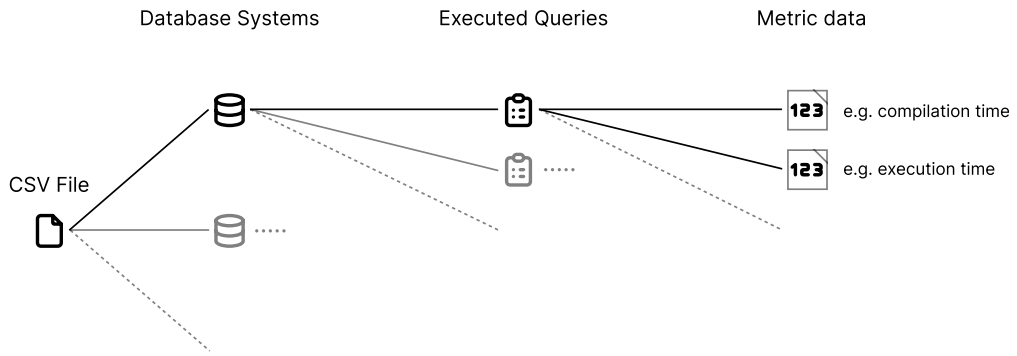


Figure 3.2: CSV structure of the input data for the Benchy Viewer

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

TODO: 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 visualization 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 analyzing 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 visualizations. Multiple views are necessary to construct a complete understanding of the current point of interest. The optimal structure of these views can 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 visualization structure for collaborative exploration.

4.1.1 Visualizing Benchmark Data

Visualizing 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 visualization. In addition to visualization, 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 colored toggles, allowing users to activate or deactivate their appearance in the visualization. We dive deeper into the interactive legend in 4.1.2.

Figure 4.1: Gobal Legend showing all participating Database Systems.

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 visualization, 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 color as the benchmark, as it processes all queries faster than all other systems. In this visualization, 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 queries.

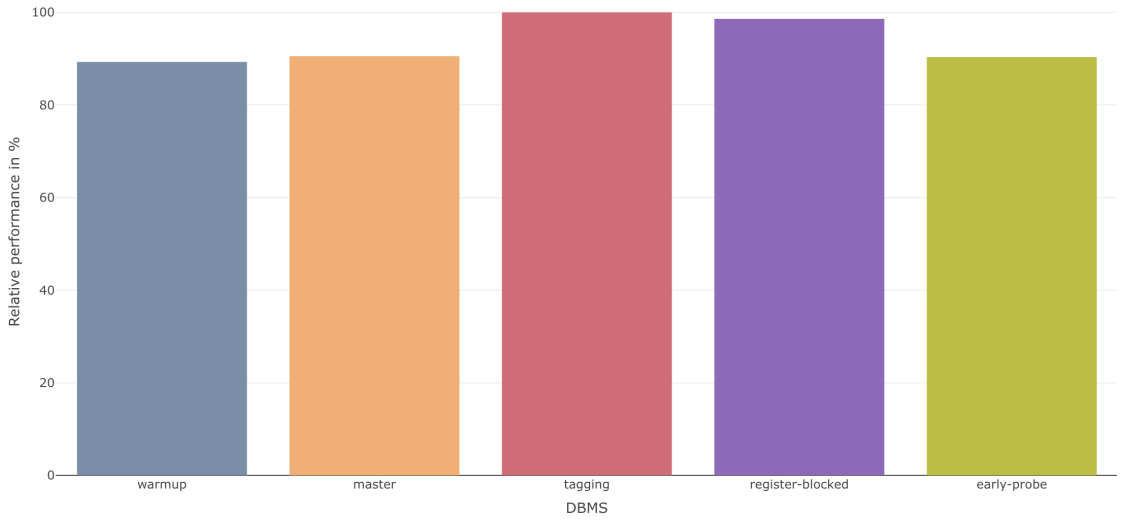


Figure 4.2: Bar chart visualizes the relative performance of all systems compared to the best performing system.

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. 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.

In this example, a comparison is made between two database systems, with the Y-Axis

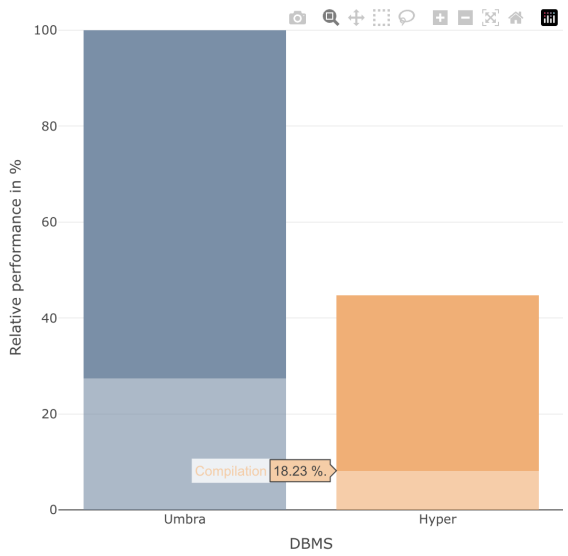


Figure 4.3: Stacked Bar Chart illustrating the Distribution of Time between Compilation and Execution steps: The compilation step is depicted in a transparent accent color, while the execution step is represented in the full color intensity.

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 visualization solution Referenz auf PDF shows essential benchmark visualizations, 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.

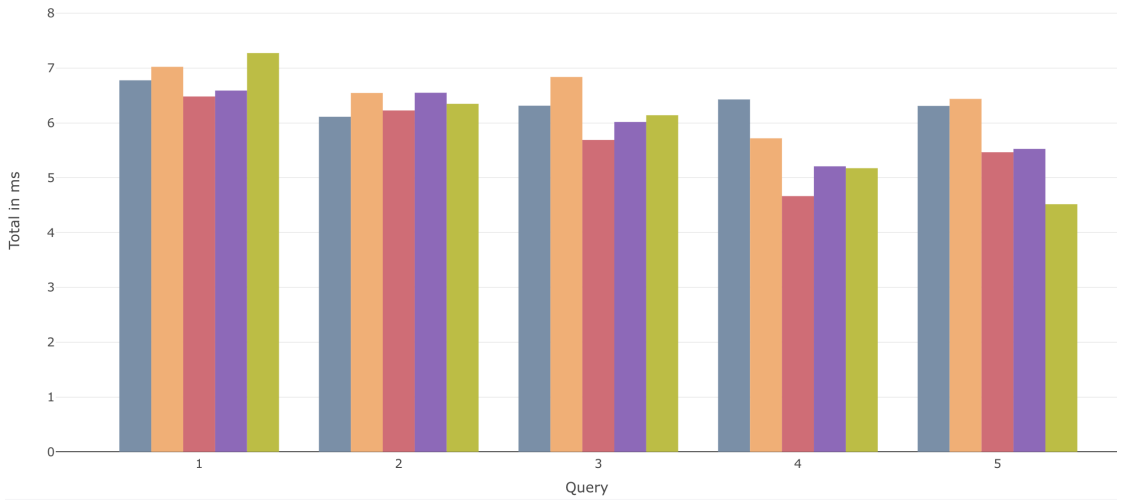


Figure 4.4: Bar chart visualizes the totals time in ms of different queries.

Bar plots stand out as a versatile visualization 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.
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 boxplots 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.

Conducting benchmark performance analysis often necessitates a comparative approach between a chosen system and other competing systems. The Benchy Viewer facilitates

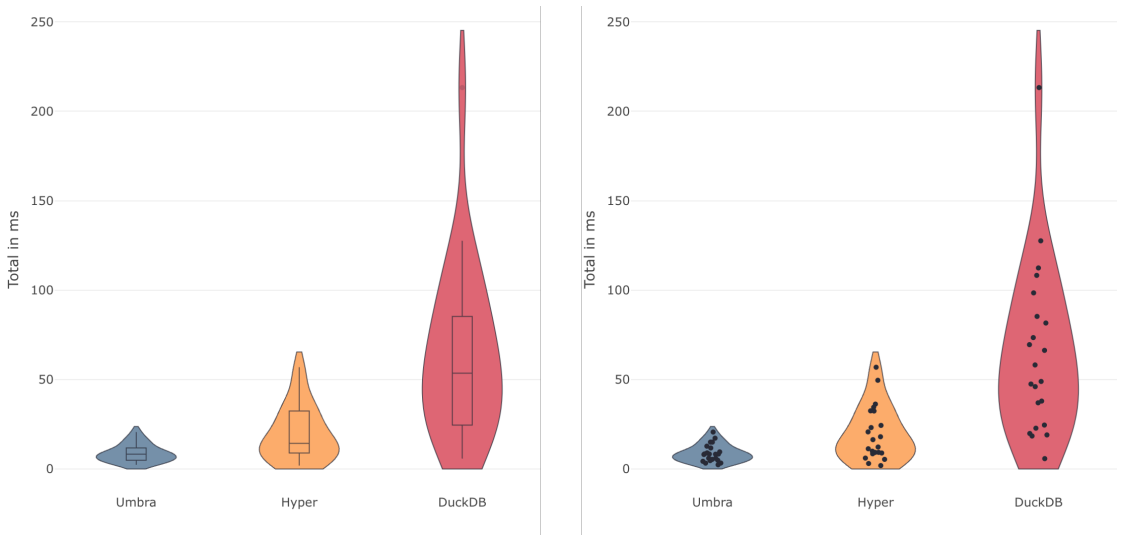


Figure 4.5: Violin charts visualize the totals time in ms of different queries. The left variant contains a boxplot and the right variant contains all data points.

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.

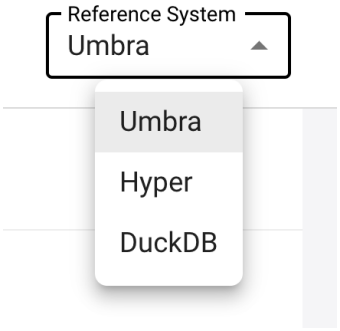


Figure 4.6: A drop-down menu that allows to select a baseline system.

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 visualizations in the application utilize the baseline system, including the table view and scatter plot for inspecting the slowdown and speedup metric, and the bar chart showcasing the performance delta for each

query from the perspective of the baseline system. All these visualizations 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 slowdown and maximum speedup become crucial. They offer a comprehensive view of the range of performance variations. Maximum slowdown indicates the worst-case scenario of reduced performance, while maximum speedup highlights the most significant improvement achieved.

Slowdown indicates how much faster 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 slowdown value greater than 1 implies that the system is slower than the baseline. For example, a slowdown of 1.5 means the system is 1.5 times slower than the baseline.
Identifying slowdowns 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 slower a specific system is compared to the baseline system. It is calculated similarly to slowdown 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 slowdown and maximum speedup, visualized in Figure 4.7, where the maximum slowdown is displayed on the left side and the maximum speedup on the right side using Umbra as the baseline system. The use of cell colors in the table serves as an effective visual indicator of performance outliers. Intensity in color corresponds to the extent of the outlier, offering a rapid understanding of the range and distribution between these extreme values. This color-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 slowdown 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 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

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 slowdown and the maximum speedup using color intensity to indicate performance outliers.

performance differences in either scenario.

The scatter plot is another effective choice for visualizing speedup and slowdown. 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 colored 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 visualizes 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 visualization occurs, highlighting the performance 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

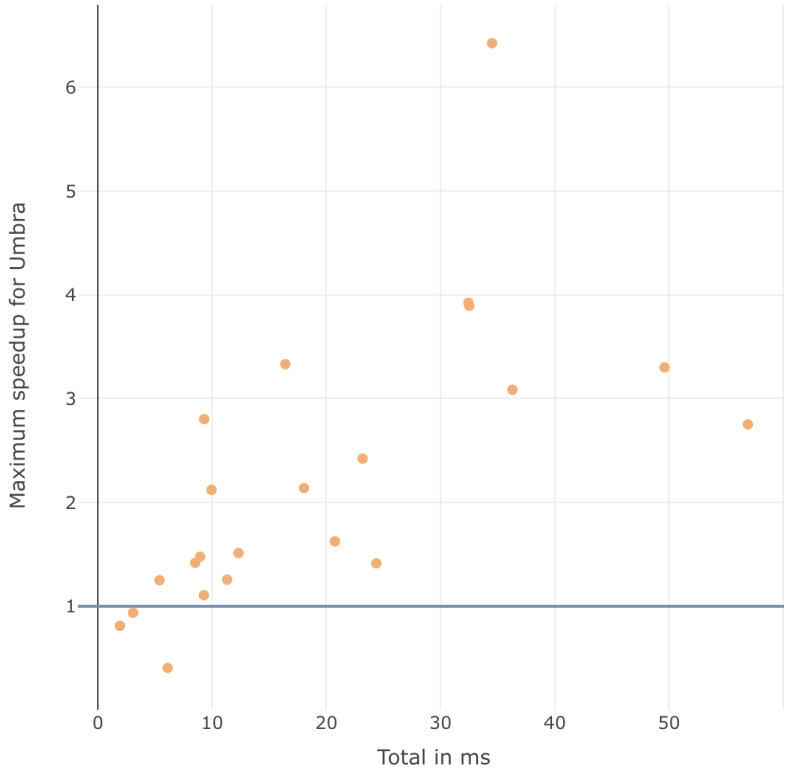


Figure 4.8: Scatter Plot visualizes the speedup.

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 colors, 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 delta for the baseline system. Additionally, the bar of each query is marked with the color of the corresponding best-performing database system.

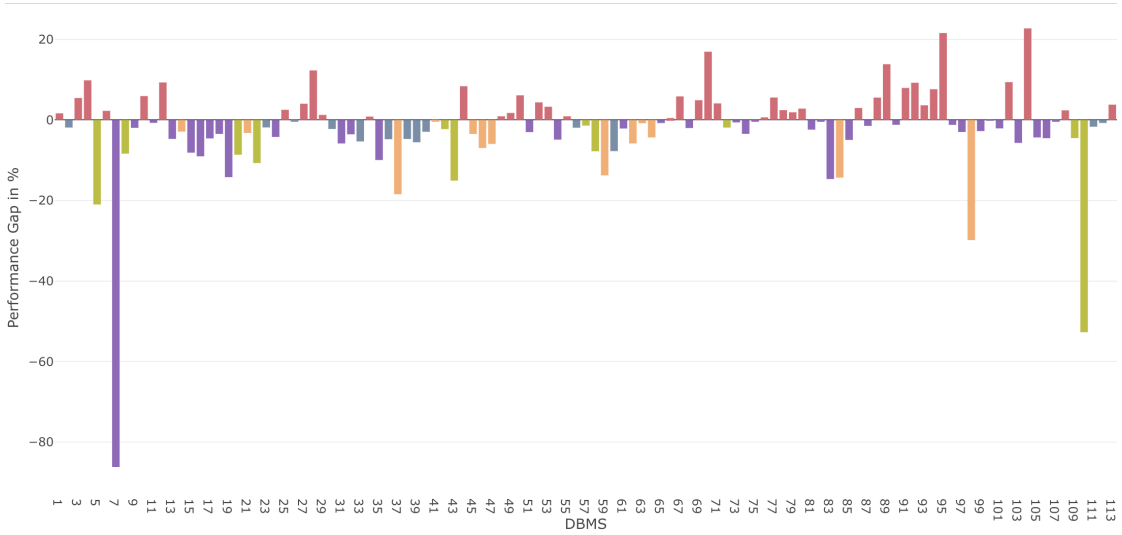


Figure 4.9: Bar chart visualizes the performance gap for every query of the baseline system compared to best corresponding query of the competing systems.

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 visualization 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 analyzing query performances and identifying noteworthy queries within visualizations, 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 a global hover capability within the application.
This means that when a user hovers over a specific query, not only is that query

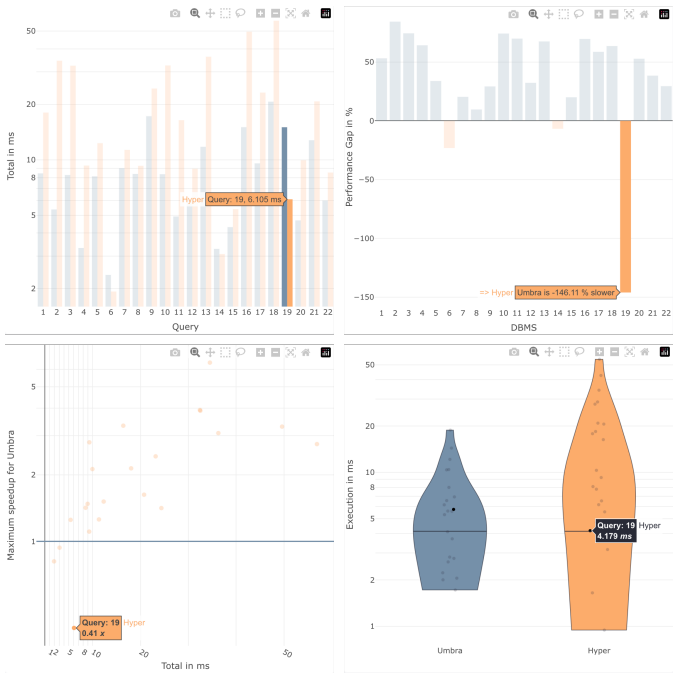


Figure 4.10: Hovering over a query automatically highlights in a global context the same query in all visualizations which are showcasing single queries. These visualizations include violin plots, scatter plots, and bar charts when one bar represents a single query.

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. With this feature, the user is able to easily identify the same queries across various visualizations, offering a comprehensive understanding from diverse perspectives.

Data Viewer

In addition to visualizations, 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 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

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 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.

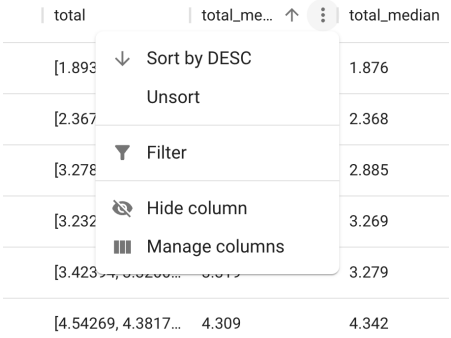
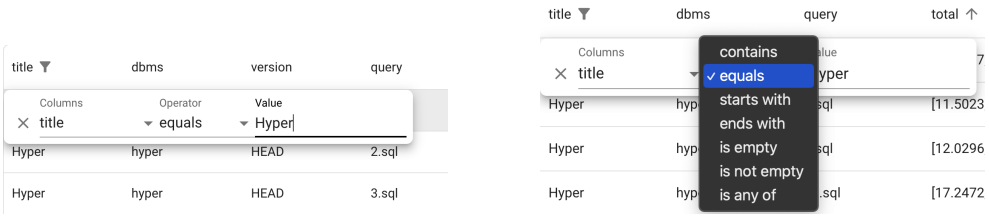


Figure 4.12: Column options drop-down offering sorting, filtering, and column visibility functionality.

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



(a) Filter Menu. (b) The list of available operators.

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

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

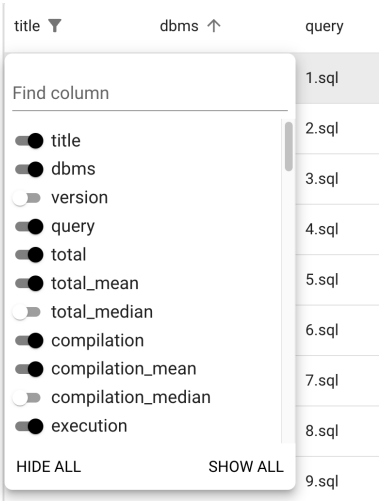


Figure 4.14: Column Manager: Easily control column visibility in the data sheet for a tailored view.

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.

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.

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

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 based on current needs.

In the course of analyzing 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.

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

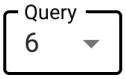


Figure 4.16: Drop-down for selecting a specific query for deeper inspection.

drop-down menu, as illustrated in Figure 4.16, or by clicking on a specific query data point within a visualization (e.g., clicking on a specific bar in a bar chart or a specific data point in a scatter plot).

Choosing a specific query results in its highlighting across all visualizations, 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.

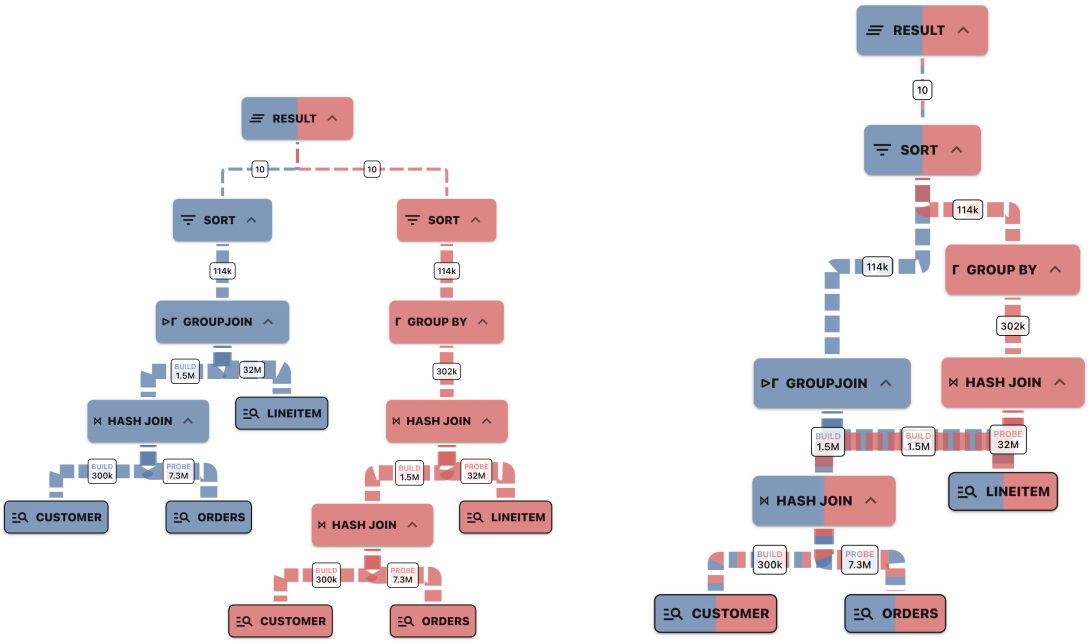
Comprehensive Query Insights

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 color. 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.

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



(a) Query Plan View comparing the query plan of a specific query between two database systems. The trees are displayed separately.

(b) Comparison of different query plans, where similar parts are merged together.

Figure 4.17: Query plan comparison between different database systems.

queries, as varying query plan structures are more clearly visualized. For an in-depth exploration of merging strategies, please refer to Section 4.5.

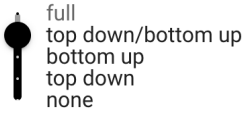


Figure 4.18: Selection of the query plan merging strategy in the form of a slider.

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.

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

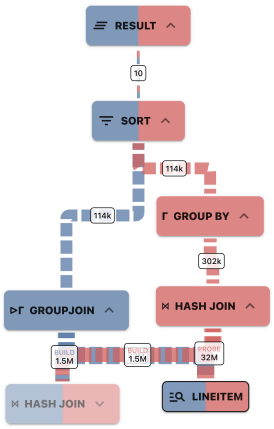


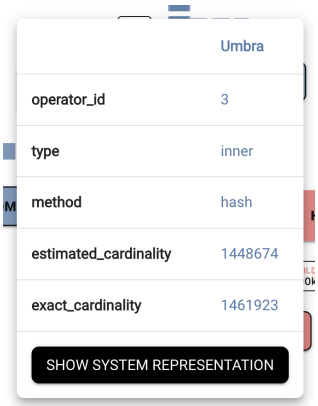
Figure 4.19: Query plan with a hidden subtree in the bottom left corner.

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.

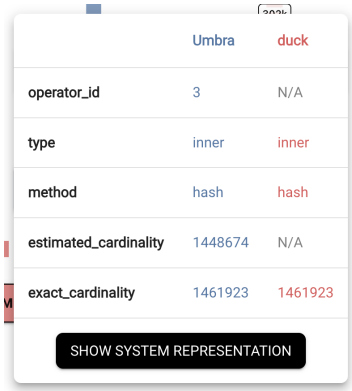
Figure 4.20: Map of the query plan showing the current location and the tree in a simplified version.

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, 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.

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



(a) Node information window.



(b) Node information window of a shared node.

Figure 4.21: Node information window of the selected node.

on the left, and an overview of the file is shown on the right.

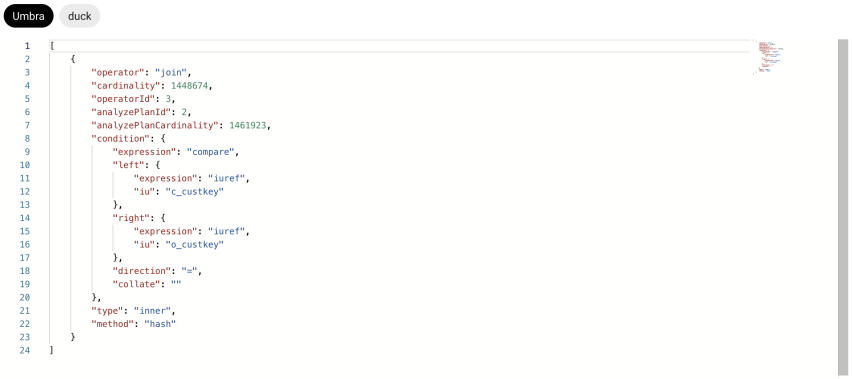


Figure 4.22: Query plan system representaion of a node.

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 standalone application Query Plan Visualizer [23k]. We dive deeper into the integration of the semantic diff tool in the section 4.5.

4.1.3 Flexible Interface Hub

To offer a well-suited platform for analyzing 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 visualizations and construct a comprehensive overview.

Subsequently, we will delve deeper into the actual visualizations, 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.

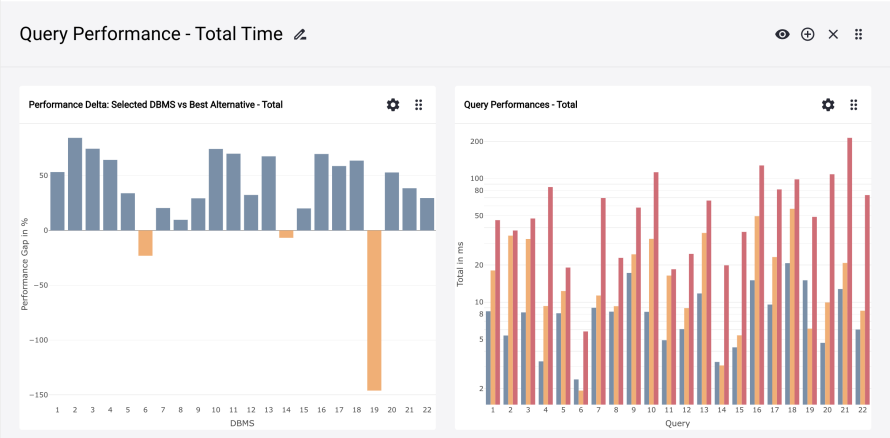


Figure 4.23: Flexible interface hub through drag-and-drop feature for visualization elements and their containers.

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 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.

Figure 4.24: Text input for renaming containers.

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

This intuitive process streamlines the augmentation of analytical sections, allowing users to dynamically enhance their workspace with relevant visualizations. 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

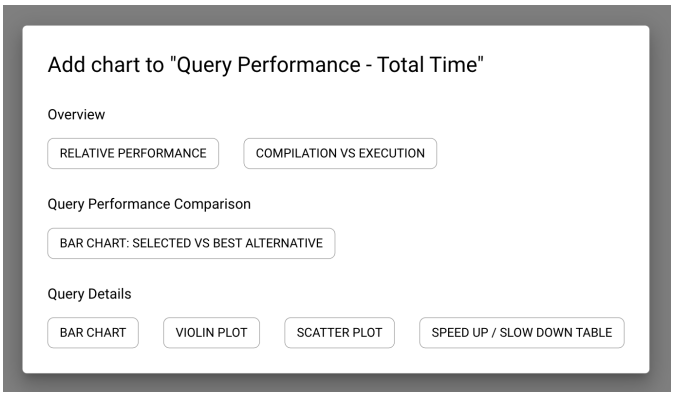


Figure 4.25: Overlay for adding charts to the current container.

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.

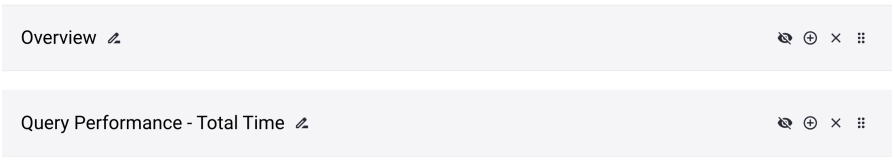


Figure 4.26: Visibility property to hide or expand a container.

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 visualizations. Delving into individual visualization 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 visualization in the context of maximum speedup and maximum slowdown, and finally, highlight global chart options.

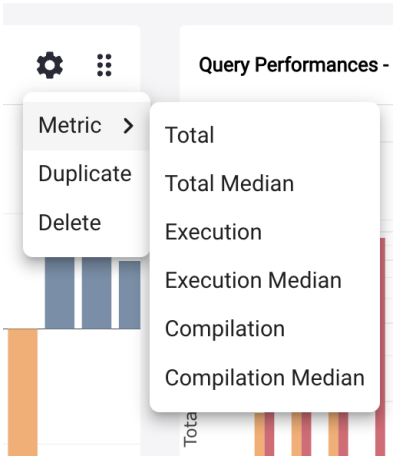


Figure 4.27: Drop-down menu of chart options offering to select the visualized metric. Also the delete and duplicate functionality is provided.

Every visualization 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 visualization 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 speedup and slowdown, seamlessly toggling between maximum speedup and maximum slowdown 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.

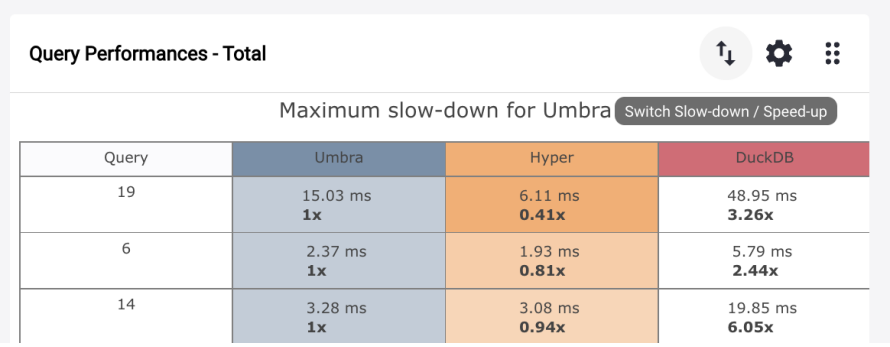


Figure 4.28: Table visualization provides the functionality to switch between maximum speedup and maximum slowdown.

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 speedup and slowdown 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.

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.

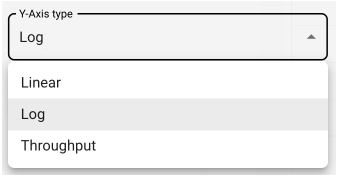
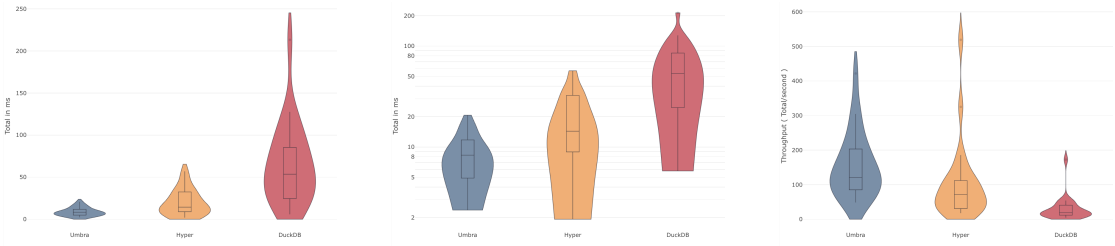


Figure 4.29: Drop-down menu of global options offering the selection of the scale type.



(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.

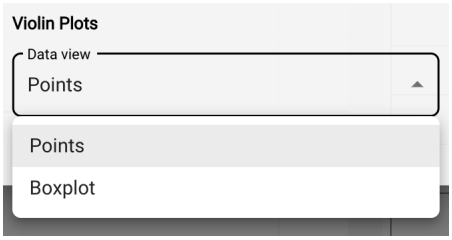


Figure 4.31: Drop-down menu of global options, offering the selection of displaying data points or a boxplot within the violins of the violin charts.

4.1.4 Saving and Sharing the Application State

One of the valuable features of the Benchy Viewer is its ability to save and share the application state. This encompasses all configurations related to visualization elements, global options, and dashboard settings. Users can conveniently save their current setup or download it for future reference, as depicted in Figure 4.32.

Downloading Configuration

The saved configurations can be downloaded as a file, allowing users to keep a record of specific setups or share them with colleagues. This downloaded file serves as a snapshot of the application state at the time of saving.

Uploading Configurations

To recreate a previous analysis session, users can upload a saved configuration file. This action loads all the configurations, restoring the dashboard layout, applied visualizations, and global settings. It's a time-saving feature that ensures a seamless transition back to a specific analytical context.

Facilitating Collaboration

The ability to share configuration files promotes collaboration among users. Team members can easily exchange analysis setups, ensuring a consistent view of data and enabling more effective collaboration on complex analytical projects.

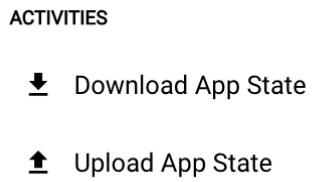


Figure 4.32: Downloading and uploading the application state.

In essence, the "Saving and Sharing" feature enhances the flexibility and collaborative potential of the Benchy Viewer, providing users with a convenient way to exchange analytical contexts.

4.2 Design

In this section, we explore the foundational principles guiding the Benchy Viewer's user interface. Rooted in Material Design, the application employs Material UI components for a clean and familiar look. The deliberate color scheme, restricted to black, white, and gray accents, promotes simplicity.

Moving on, we delve into the sidebar and the header, ensuring constant visibility and facilitating seamless navigation. The sidebar acts as a central hub, offering functionalities from page navigation to data import. The header contains crucial elements like the legend, eliminating the need for one in each visualization.

Finally, we examine the application's pages: Analytics Dashboard, Query Plan View, and Input File View. Each serves a distinct purpose, from analysing benchmark data to offering a minimalist view of raw imported data.

4.2.1 User Interface Design Styles

Material UI

The Benchy Viewer's design is rooted in the principles of Material Design [23c], emphasizing a sleek and consistent user interface that aligns with modern design standards. To ensure a cohesive and user-centric design, the application makes extensive use of predefined components from Material UI [23d]. In our application, this includes buttons, inputs, toggles, sliders, data grids, icons, drop-down menus, and tooltips.

The inclusion of Material Design components contributes to an interface that is both user-friendly and familiar. Users can easily navigate through these components, thanks to the standardized styling that Material UI provides.

Icons within the Benchy Viewer serve as visual cues, representing familiar symbols that aid users in quickly grasping the functions they perform. This visual clarity aligns with Material Design's emphasis on intuitive iconography.

Settings options, presented in drop-down menus and other interfaces, follow Material Design practices. The design prioritizes user intuition, allowing individuals to interact with the application seamlessly without relying on extensive textual explanations.

By aligning with Material Design principles, the Benchy Viewer achieves a design that not only meets aesthetic standards, but also prioritizes user understanding and interaction efficiency. The incorporation of familiar components enhances the overall usability and accessibility of the application.

Styling Characteristics

The Benchy Viewer embraces a deliberate color scheme aimed at providing a clean design. This is achieved through a restraint to specific base colors, limiting the palette to black, white, and a gray accent color, as depicted in Figure 4.33. By adhering to this minimalist approach, the application exudes a sense of simplicity, ensuring a visually uncluttered interface.

Figure 4.33: Color palette of the user interface.

While the overall design adheres to a subdued color palette, the visual elements within the application utilize a distinct color scheme, as illustrated in Figure 4.34. This strategic approach guarantees that charts, plots, and other visual elements command attention, supporting users to focus on those data visualizations.

To enhance user experience, the Benchy Viewer incorporates immediate feedback mechanisms. When users hover over interactive elements, such as buttons or inputs,

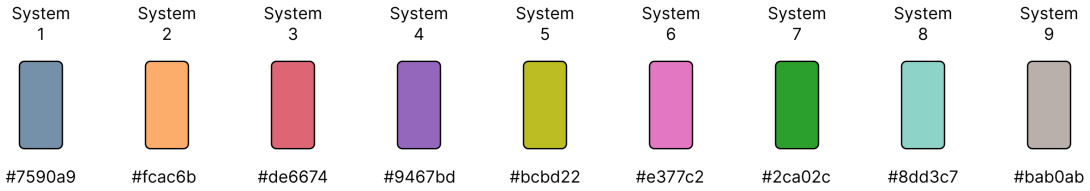


Figure 4.34: Color palette used by visualizations.

accent colors dynamically adjust, providing a visual cue of the interactive nature of the element. Additionally, the mouse representation undergoes subtle changes, reinforcing the responsiveness of the interface.
Additionally, the Benchy Viewer employs the Roboto font [Goo23] for a clean and modern look, enhancing readability and contributing to a user-friendly interface.

4.2.2 Page Structure and Navigation

The design of the Benchy Viewer is thoughtfully crafted for user convenience and simplicity. This section provides insights into the structural components that define the organization of the Benchy Viewer.

At the heart of the application's architecture are the header and the sidebar, ensuring constant visibility for users and facilitating seamless navigation. These elements contribute to maintaining a strong sense of orientation throughout the user journey.

Sidebar

The Benchy Viewer adopts a standard web view with a sidebar, leveraging the familiar layout seen in many web applications, as illustrated in Figure 4.35. This approach comes with inherent advantages. Users accustomed to web interfaces will find this structure intuitive and easily navigable, contributing to a seamless user experience.

The sidebar serves as the central hub for navigating different sections of the application, while always staying present. By providing a navigation button section, users can effortlessly transition between pages, ensuring quick access to specific functionalities and data views. Additionally, the navigation button of the current page is highlighted with the accent color.

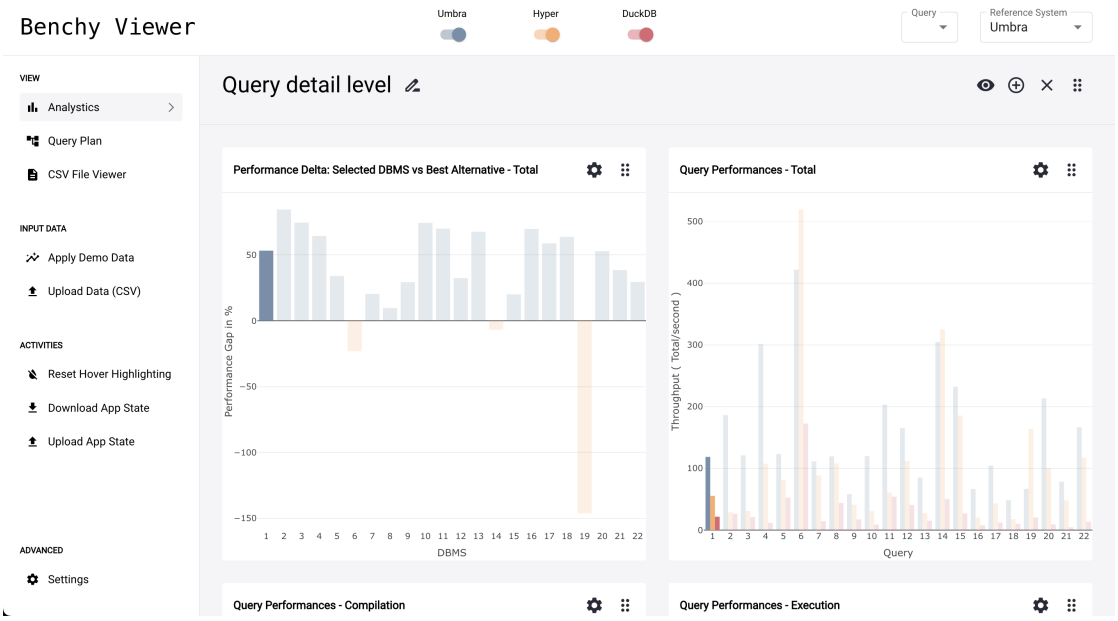


Figure 4.35: Overview of the application structure.

Below, the users have the possibility to import benchmark data or explore the application's capabilities using demo data. This feature facilitates a hands-on experience without the immediate need to provide personal datasets, making it convenient for users to evaluate the application's functionalities.

The sidebar offers additional functionalities, including the ability to reset hover highlighting. Users can download and upload the application state, enabling them to save configurations and share them or reload them for future sessions, which is further examined in 4.1.4.

A settings icon button in the sidebar provides direct access to the global settings of the Benchy Viewer, allowing users to adjust application-wide settings from anywhere within the application.

Header

The header, along with the sidebar, remains visible in all scenarios within the Benchy Viewer. While the sidebar offers a range of functionalities, the header contains fewer elements. The central element of the header is the legend, strategically placed to eliminate the need for a legend in each individual visualization, promoting a clean and

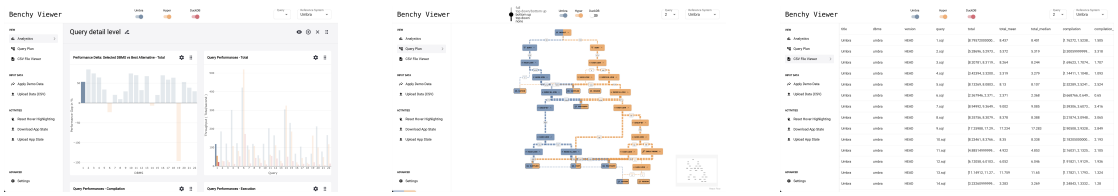
efficient overview. Users can activate or deactivate any system at any time using the toggles within the legend.

Another crucial functionality in the header is the selection of the baseline system and the focused query, offering the same advantage of accessibility at any point in the user's workflow.

Additionally, the header adapts contextually when navigating to the "Query Plan" page. In this scenario, a slider appears for selecting the comparison strategy between query plans, providing further flexibility and control, as explored in Section 4.5.

Pages

The Sidebar and Header form the foundation of the user interface, while the remaining space is dedicated to displaying the various pages. Essentially, the Benchy Viewer comprises three main pages, depicted in Figure 4.36: the Analytics Dashboard page, the Query Plan page, and the Input File Viewer page.



(a) Analytics Dashboard. (b) Query Plan View. (c) Input File Viewer.

Figure 4.36: From left to right: Analytics Dashboard page, Query Plan View page, and Input File Viewer page.

The Analytics Dashboard page features the drag-and-drop system for visualizing elements and their containers. This page serves as the hub for analysing benchmark data, offering charts and plots to explore queries from diverse perspectives.

Upon identifying significant queries using the analytics dashboard, users often transition to the Query Plan page. This page serves as a central hub for comparing distinct query plans across various database systems.

In contrast, the Input File Viewer page offers a minimalist perspective. Tailored for users seeking an unembellished view of their imported benchmark data, this page omits visualizations. It provides an in-depth examination of the raw data, enabling users to scrutinize the dataset's structure and intricacies without the influence of charts or plots.

4.3 Data Structure

Unraveling the complexities of the Benchy Viewer necessitates a deep dive into its fundamental data structure, a cornerstone shaping the app's functionality and user interaction.
Our exploration commences with an examination of the broad project data structure, leading to a comprehensive exploration of each intricate facet of the global application state.

4.3.1 Overall Project Structure

The Benchy Viewer embodies a robust project structure that leverages React, global state management, and a page navigating router, as illustrated in Figure 4.37, which we will explore in this section.

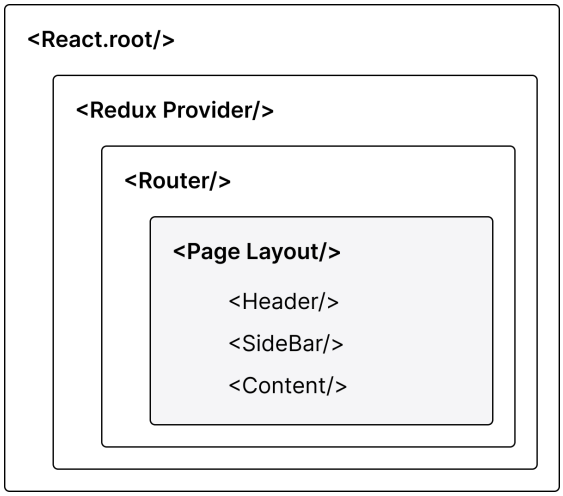


Figure 4.37: Project Structure Hierarchy.

Root Component

The root component is the entry point of the Benchy Viewer React application, orchestrating critical processes that define its robustness and efficiency.
The primary responsibility of the root component is to render the main application, serving as the nexus for various functionalities, providing the structural and organizational backbone for the entire application.

In the project structure hierarchy, the root component encapsulates the application with `<React.StrictMode/>` [23i], which is a tool that highlights common issues and potential problems in a React application during development. In general, it detects impure calculations in the context of component states and multiple rendering. It enables a set of additional checks and warnings to catch and alert developers about unsafe or deprecated practices, contributing to better code quality. While it doesn't affect the production build, it's a valuable aid in identifying and addressing issues early in the development phase.

State Management with Redux

The Redux Provider in the Benchy Viewer, built using Redux [Abr23], serves as the central hub for managing the application's state. It acts as a shared space where different parts of the application can store and retrieve data efficiently.
Utilizing the capabilities of *Redux Toolkit* [AR23], the Redux Provider supplies the global state across the entire application. This global state is compartmentalized into five distinct slices, illustrated in Figure 4.38, each serving a unique purpose.

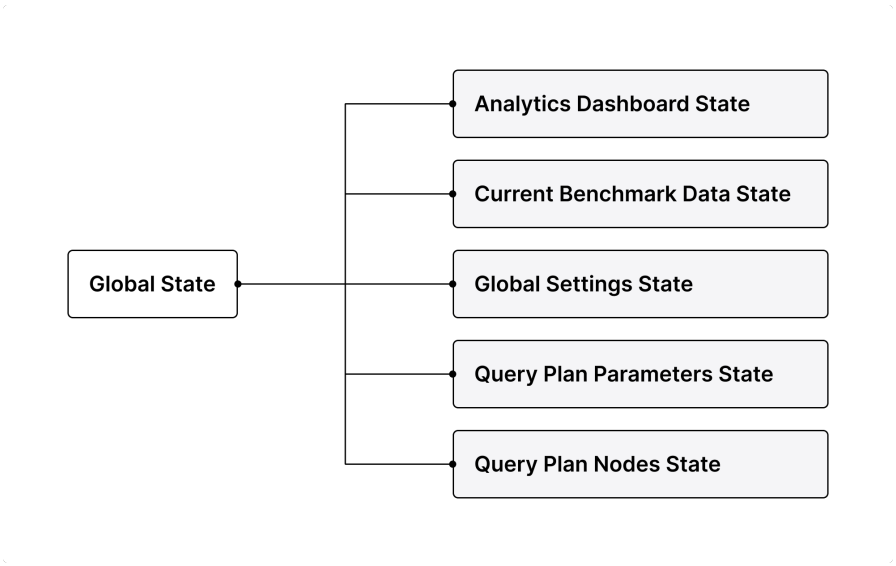


Figure 4.38: Application State Compartements.

The Analytics Dashboard State encapsulates all the data concerning visualizations within the Analytics Dashboard. It includes information about the position within the dashboard and the configuration of each visualization element. A more detailed exploration of this data structure is provided in Section 4.3.4.

The Current Benchmark Data State is responsible for holding all benchmark data from various database systems. This state is a pivotal component used across the Analytics Dashboard, Query Plan View, and Benchmark Data View. A detailed examination of this state's structure is available in Section 4.3.2.

Focused on interactive features within visualizations, the Global Settings State encompasses functionalities like the global hover state or the selected query. Section 4.3.3 delves into the specifics of this data structure.

Tailoring the visualization needs for the user, the Query Plan Parameters State contains data related to the appearance and comparison types of the query plans. More insights into this state are provided in Section 4.3.5.

Lastly, the Query Plan Nodes State unfolds the story of the actual query plan, housing details about every node and their associated information. For a comprehensive exploration, turn to Section 4.3.5.

Router

Navigating through the corridors of the Benchy Viewer's project structure, the router, powered by *React Router* [Inc23], providing a standardized approach to handle navigation in React. Specifically, in our application, this router guides users through three key areas: the Analytics Dashboard, Query Plan View, and Data Viewer.

Layout and Content

Finally, we encounter the pages with their contents within the Benchy Viewer. These pages share a consistent structure, ensuring a unified user experience. Each page adheres to a standardized layout, featuring a sidebar, a header, and a dedicated space for content, which we discussed the page structure in 4.2.2.

At this point, the various components at different levels come together to form the Benchy Viewer application. The root component lays the groundwork, overseeing essential processes. The Redux Provider acts as a central hub for sharing information across the application. The global state, divided into five slices, provides state to all pages. The router facilitates easy navigation between pages, which is controlled in the sidebar. In essence, these pages, with their structured layouts, unite to offer users a seamless and comprehensive experience.

4.3.2 Benchmark Data

The initial step when engaging with the Benchy Viewer involves importing a file containing performance data slated for visualization. This file must adhere to the format outlined in Section 3.2.2 and is processed by the Current Benchmark Data State, introduced in the previous section.

Import Process

Throughout the importation process, the input file undergoes parsing and transformation into a TypeScript object. Subsequently, the benchmark data is partitioned according to the respective database systems. This partitioning ensures that all query data aligns with its associated database system. The resulting data structure encompasses a system ID, a title, all pertinent benchmark data, and an activation flag.

Data Accessibility

After the data import, the benchmark data is accessible to all visualization elements across the whole application. This includes all visualization elements within the Analytics Dashboard, the Input File Viewer, and the Query Plan View.

The activation flag controls the visibility of the system data of a whole database system in the visualizations. This is accessed through the header's legend toggles, where users can control the activation or deactivation of database systems.

Additionally, this state holds the demo data, which is initialised on the start of the application. Upon user requests via the sidebar, the demo data is loaded as the current benchmark data.

4.3.3 Global Settings

In the Benchy Viewer, the global settings serve as a central hub for interactive and customizable data exploration. Features like hover and selected queries offer real-time insights, while selecting a baseline system deepens understanding for comparative analysis. Visual configurations provide the power to mould data representation, aligning with user preferences. In essence, the global settings act as the control center, empowering users for dynamic and tailored data exploration.

Hovered Query

A key interactive feature is the global hover functionality, where details about the currently hovered query are stored. This information is triggered when a user hovers over a specific query within a visualization element on a detailed level, like a bar in a bar chart. This hover information is then synchronized across all other visualizations simultaneously.

Selected Query

Much like the hover feature, the selected query is stored and activated through interactions within the visualization elements at a detailed level. Additionally, a drop-down input in the header allows users to manually select the current query. This state is utilized not only for highlighting purposes in the Analytics Dashboard, but also in the Query Plan View for selecting the corresponding query plan.

Baseline System

Another crucial component within this data slice is the selected baseline system, a linchpin for comparative analyses in the Benchy Viewer. This feature empowers users to delve into the results of their chosen system, comparing and contrasting its performance against other systems. This is particularly valuable in visualizations incorporating metrics like speed-up and slow-down.

Specifically utilized in visualizations such as the comparative query bar chart, the selected baseline system plays a vital role in illustrating the performance delta for each query. This allows users to discern the nuanced differences between the baseline system and the best-performing alternative.

Visualization Configurations

This segment houses configuration options influencing the visual aspects of charts, detailed in Section 4.1.3. It governs factors like the chart axis scale and display type for violin plots. Users can conveniently access and tweak these options through the settings menu in the sidebar. The choices made here resonate across various visualization elements in the Analytics Dashboard, providing a tailored and cohesive visual experience.

4.3.4 Analytics Dashboard Data Structure

The Analytics Dashboard in the Benchy Viewer acts as a central hub for comprehensive performance analysis, providing a structured and user-friendly interface. At its core, the data structure slice forms the backbone, storing essential information about analytics sections and the visual elements they contain.

Containers

This data slice manages a roster of analytics sections, with their order mirroring the sequence in the UI's drag-and-drop system. Reordering the analytics sections in the UI through the drag-and-drop feature will update this state.

Each container in the list retains a name for labeling the analytics section. Additionally, a visibility flag dictates whether the container is collapsed or its content, along with all its charts, is visible. In general, all of these properties are accessible in the UI through the containers header.
Lastly, every container holds a list of its associated visualization elements.

Visualization Element

Within each container, a list of visualization elements is stored, aligning with the sequence in the UI's drag-and-drop system. Reordering charts in the UI through the drag-and-drop feature will update this state.

These visualization elements are distinguished by a label in their header, signifying their content.
Furthermore, each element is characterized by a property denoting its visual type, such as a bar chart or violin plot.
Lastly, a vital attribute is the metric associated with each visualization element, such as execution time or compilation time. In the UI, this is accessible through the chart menu, see 4.1.3.

4.3.5 Query Plan

The incorporation of the query plan into the Benchy Viewer relies heavily on the integration of the Query Plan Visualizer [23k]. For a more detailed exploration of this integration, refer to Section 4.5.

The query plan's data structure is divided into two essential parts. First, there are parameters governing the visual aspects and comparison strategies of the query plans. Second, we have the actual query plan data for the database systems.

Visualization Parameters

To enhance the user experience, the Query Plan Viewer incorporates a collapsible functionality in every node, denoted by an icon. The activation of this functionality at a specific node will collabse its whole subtree.
Recognizing that some users may find this feature unnecessary, an option exists in the settings to disable collapsibility. This choice hides all icons in the nodes, ensuring a cleaner and less distracting look for the query plan.

The matching algorithm parameter is a pivotal component influencing the visualization of query plans in the Benchy Viewer. This property, structured as an enum, provides a spectrum of options to tailor the matching algorithm based on specific visualization needs. It is accessible in the header when the Query Plan View is active, as depicted in Figure 4.18. When set to "none", it allows query plans to be displayed independently, side by side, without any comparison. Alternatively, choosing "top down" or "bottom up" initiates matching algorithms that prioritize either a top-down or bottom-up approach, optimizing clarity in specific scenarios. The "top down/bottom up" option combines both methods, offering a balanced approach.

Lastly, users have the option to customize the visualization of edges within the query plan. By default, edges are presented in horizontal and vertical directions, reminiscent of a classical tournament tree. However, when dealing with intricate query plans or non-planar graphs, overlapping edges may compromise clarity. To address this, users can modify the edge style. This customization allows edges to deviate from strict horizontal or vertical orientations, adopting a more dynamic style that allocates additional space to each edge. This flexibility ensures that the visual representation of edges aligns with the structure of the query plans, contributing to a more informative visualization.

Query Plan Data Structure

During the import of benchmark data into the Benchy Viewer, the query plan metadata undergoes a transformation process to populate a structured object. This involves parsing the raw metadata provided in the benchmark results, which includes details such as the operator ID, estimated cardinality, and exact cardinality.

The data transformation involves creating instances that represent each element of the query plan, forming a tree structure to reflect the hierarchical arrangement. This establishes relationships between elements to capture the flow and nesting of operations. System-specific details are retained within each node throughout this transformation. This structured representation enables users to navigate the intricacies of the query execution flow.

4.4 Integration of Plotly-React for Data Visualization

For visualizing the data within plots and charts, the graphing library *Plotly* [23h], was used. Plotly is a versatile and interactive graphing library widely used for data visualization. Its compatibility with languages like JavaScript, through *Plotly JavaScript* [23g], enhances its appeal for web-based applications. With its features and user-friendly interface, Plotly is a solid choice for rendering interactive visual representations of data.

4.4.1 General Features

Plotly comes out of the box with a bunch of features that align with the interactability of the Benchy Viewer.

Plotly allows users to effortlessly scroll, zoom, and pan around plots, offering a dynamic exploration of the data. The ability to reset zoom provides added convenience, ensuring users can navigate through intricate details with ease.
Plotly further extends its functionality by enabling users to download the current representation as a PNG file, facilitating seamless sharing and documentation of visual insights.
The interactive hover feature adds another layer of depth to the visualization, allowing users to access specific data points with a simple cursor hover. The hover feature in Plotly was used to form the global hover feature, which we will discuss further in 4.4.2.

4.4.2 Integration of the Global Hover Feature

Plotly offers great customization capabilities to developers, particularly when interacting with data points through the hover feature. When hovering over data points, Plotly provides detailed information, including the value and index. This hover information undergoes filtering, where the resulting query number is sent to the corresponding data slice.

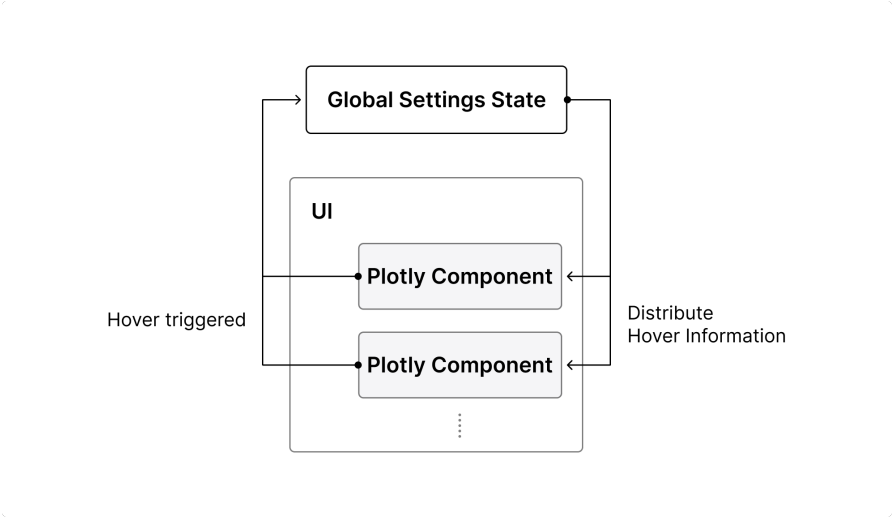


Figure 4.39: Distribution of the Hover Information.

Subsequently, the UI is automatically updated, and the hover information is distributed to all Plotly components. This dynamic process ensures that the queries associated with the provided information are highlighted, creating a synchronized and insightful data exploration experience. For a visual representation of this data flow, refer to Figure 4.39.

4.4.3 Wrapper Component providing necessary Properties

To ensure seamless integration and synchronized functionality with the Benchy Viewer, Plotly components are encapsulated within a specialized wrapper component. This wrapper, illustrated in Figure 4.40, incorporates essential features.

Firstly, it constructs the chart header, facilitating a drag-and-drop interface that seamlessly aligns with the Analytics Dashboard's drag-and-drop functionality. The header also encompasses chart settings, including a nested menu where users can select metrics and perform basic actions like duplication or deletion. Additionally, a descriptive label elucidates the content of the visualization element.

All the information presented in the wrapper is orchestrated by the controller, which manages state changes, such as when a user selects a different metric in the chart settings.
The controller is also responsible for preparing the data for visualization which involves the conversion of benchmark data into a data structure compatible with Plotly. This process may be divided into multiple stages, especially when utilizing metrics like

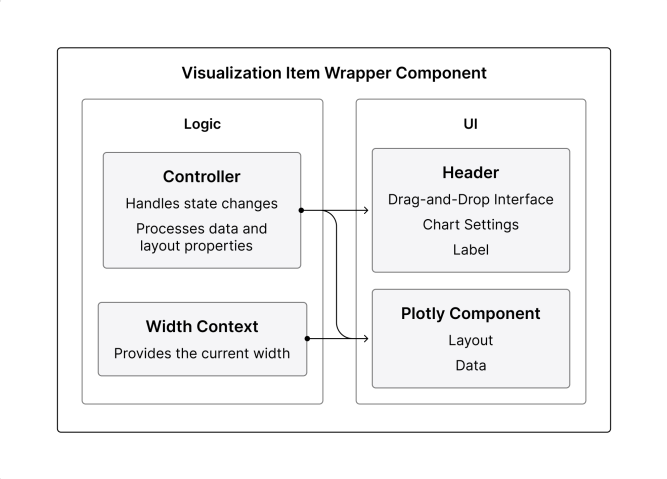


Figure 4.40: Wrapper Component Structure.

speed-up and slow-down, which require additional calculations before presenting data to Plotly.
Calculations for proper scaling on the x-axis and y-axis are performed, taking into account the dataset's highest values. Furthermore, the controller constructs appropriate axis properties, such as the proper tick values, based on the visualization type and chart settings.

A notable consideration is that Plotly components expect concrete values for width and height. To accommodate varying screen sizes and dynamic resizing, the Benchy Viewer employs the React concept *createContext* API [23a] to form the `Width Context`. The implemented mechanism monitors the current width of the component and the `Width Context` propagates it down the component hierarchy. The mechanism, attuned to window resizing events, ensures that all components using the width prop from the `Width Context` are updated whenever the window size changes. This dynamic approach enables Plotly components to adapt effectively to different screen sizes and resizing scenarios.

4.5 Integration of the Query Plan Visualizer

Integrating the Query Plan Visualizer into the Benchy Viewer presented a challenge due to differing global state management systems. The Query Plan Visualizer uses React Sweet State [23j], while the Benchy Viewer relies on Redux [Abr23].
In the Business Logic subsection, we discuss the conversion process from React Sweet

State to Redux. This involved a top-level restructuring to align with Redux's global state management, while maintaining application functionality.

In the UI subsection, both tools utilize the Material UI framework [23d], facilitating seamless code reuse. We highlight the reuse of Material UI components, like sliders, and the unique features of the Benchy Viewer's header, emphasizing code consistency with the reuse of components.

4.5.1 Business Logic

The integration of the Query Plan Visualizer into the Benchy Viewer introduced a challenge due to the disparity in the global state management systems used. The Query Plan Visualizer relies on React Sweet State for managing its state, encompassing everything from the actual query plan data to various configuration settings. On the other hand, the Benchy Viewer employs Redux for managing the global state of the entire application.
To maintain coherence and prevent the use of multiple state management tools within the application, a crucial step was to convert the relevant parts of the React Sweet State to Redux. This conversion process was a key challenge given that React Sweet State and Redux, while both state management libraries for React, have distinct approaches and underlying concepts.

The conversion process from React Sweet State to Redux involved a top-level restructuring of the state management logic. In React Sweet State, the state management is more localized and hook-based, catering to the component's specific needs. On the other hand, Redux operates with a global store and relies on actions and reducers for state modifications.

To migrate, we first identified the distinct states managed by React Sweet State and refactored them into Redux actions and reducers. This required careful consideration of how data flows through the application and the structure of the Redux store. The hook-based state management in React Sweet State was replaced with action dispatches and selectors in Redux.

Overall, the conversion aimed at preserving the application's functionality while aligning with the global state management paradigm of Redux. The resulting Redux implementation provides a successfully centralized state management system.

4.5.2 UI

The integration of both the Benchy Viewer and the Query Plan Visualizer with the Material UI framework facilitates a seamless UI integration process by allowing the reuse of substantial portions of the source code. Specifically, the Benchy Viewer has effectively repurposed various Material UI components, including sliders utilized for configuring the query plan and nodes in constructing the query plan.

Despite the Query Plan Visualizer boasting an extensive header with a multitude of functionalities, the Benchy Viewer adheres to a minimalist design, featuring only the most essential operations. To maintain this design philosophy, we identified the pivotal configuration features for the query plan within the Benchy Viewer's header. Any additional configurations were logically placed in the settings menu, accessible through the sidebar. In the Query Plan Visualizer, the activation and deactivation of query plans for database systems were accomplished using a Material UI Chip component [23e]. While replicating this functionality was an option, we opted for reusing the header legend from the Analytics Dashboard, as depicted in Figure 4.41. For this functionality, the legend provides its intuitive toggles with the corresponding system color.

Figure 4.41: Legend Conversion: From the original component of the Query Plan Visualizer [23k] on the left to the interactive legend employed by the Benchy Viewer on the right.

Beyond these shared features, the Benchy Viewer's header provides the ability to control the matching algorithm, influencing the visualization of query plans. For instance, users can adopt a top-down approach to construct the query plan. The original tool implemented this with a Material UI Slider [23f], and to maintain code consistency and streamline development, we chose to employ the same Material UI Slider, thereby reusing significant portions of the source code associated with this particular functionality within the 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 Comparison with Existing Solutions

5.2.4 Potentials and Future Improvements

6 Conclusion

6.1 Summary of Results

6.2 Future Developments and Enhancements

Abbreviations

List of Figures

List of Tables

Bibliography

[23a] *createContext API*. Online; accessed on 20-November-2023. 2023. URL: https://react.dev/reference/react/createContext.

[23b] *Intel Turbo-Boost*. Online; accessed on 03-December-2023. 2023. URL: https://www.intel.de/content/www/de/de/gaming/resources/turbo-boost.html.

[23c] *Material Design*. Online; accessed on 22-November-2023. 2023. URL: https://m3.material.io/.

[23d] *Material UI*. Online; accessed on 14-September-2023. 2023. URL: https://mui.com/.

[23e] *MUI Chip*. Online; accessed on 27-November-2023. 2023. URL: https://mui.com/material-ui/react-chip/#clickable.

[23f] *MUI Slider*. Online; accessed on 27-November-2023. 2023. URL: https://mui.com/material-ui/react-slider/.

[23g] *Plotly JavaScript Open Source Graphing Library*. Online; accessed on 20-November-2023. 2023. URL: https://plotly.com/javascript/.

[23h] *Plotly Open Source Graphing Libraries*. Online; accessed on 20-November-2023. 2023. URL: https://plotly.com/graphing-libraries/.

[23i] *React Strict Mode*. Online; accessed on 17-November-2023. 2023. URL: https://react.dev/learn/keeping-components-pure#detecting-impure-calculations-with-strict-mode.

[23j] *React Sweet State*. Online; accessed on 27-November-2023. 2023. URL: https://github.com/atlassian/react-sweet-state.

[23k] *Semantic Diff / Query Plan Visualizer*. Online; accessed on 19-November-2023. 2023. URL: https://github.com/Toemmsche/semantic-diff.

[Abr23] C. Abramov. *Redux*. Online; accessed on 17-November-2023. 2023. URL: https://redux.js.org/.

[AR23] Abramov and the Redux Documentation Authors. *Redux Toolkit*. Online; accessed on 17-November-2023. 2023. URL: https://redux-toolkit.js.org/.

[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.

[FMT20] H. Funke, J. Mühlig, and J. Teubner. "Efficient Generation of Machine Code for Query Compilers." In: *Proceedings of the 16th International Workshop on Data Management on New Hardware*. DaMoN '20. Portland, Oregon: Association for Computing Machinery, 2020. ISBN: 9781450380249. DOI: 10.1145/3399666.3399925.

[FN21] P. Fent and T. Neumann. "A Practical Approach to Groupjoin and Nested Aggregates." In: *Proc. VLDB Endow.* 14.11 (July 2021), pp. 2383–2396. ISSN: 2150-8097. DOI: 10.14778/3476249.3476288.

[Goo23] Google. *Roboto Font*. Online; accessed on 22-November-2023. 2023. URL: https://fonts.google.com/specimen/Roboto.

[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.

[Inc23] R. S. Inc. *React Router*. Online; accessed on 18-November-2023. 2023. URL: https://reactrouter.com/en/main.

[KLN21] T. Kersten, V. Leis, and T. Neumann. "Tidy Tuples and Flying Start: fast compilation and fast execution of relational queries in Umbra." In: *The VLDB Journal* 30 (2021), pp. 883–905.

[Neu11] T. Neumann. "Efficiently Compiling Efficient Query Plans for Modern Hardware." In: *Proc. VLDB Endow.* 4.9 (June 2011), pp. 539–550. ISSN: 2150-8097. DOI: 10.14778/2002938.2002940.