# *IDEBench: A Benchmark for Interactive Data Exploration*

# IDEBench: A Benchmark for Interactive Data Exploration

Philipp Eichmann
Brown University
Providence, RI

Emanuel Zgraggen
MIT CSAIL
Cambridge, MA

Carsten Binnig
TU Darmstadt
Germany

Tim Kraska
MIT CSAIL
Cambridge, MA

## ABSTRACT

In recent years, many query processing techniques have been developed to better support interactive data exploration (IDE) of large structured datasets. To evaluate and compare database engines in terms of how well they support such workloads, experimenters have mostly used self-designed evaluation procedures rather than established benchmarks. In this paper we argue that this is due to the fact that the workloads and metrics of popular analytical benchmarks such as TPC-H or TPC-DS were designed for traditional performance reporting scenarios, and do not capture distinctive IDE characteristics. Guided by the findings of several user studies we present a new benchmark called IDEBench, designed to evaluate database engines based on common IDE workflows and metrics that matter to the end-user. We demonstrate the applicability of IDEBench through a number of experiments with five different database engines, and present and discuss our findings.

**ACM Reference Format:**
Philipp Eichmann, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. 2020. IDEBench: A Benchmark for Interactive Data Exploration. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3318464.3380574


*SIGMOD'20, June 14–19, 2020, Portland, OR, USA*

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3380574

## 1 INTRODUCTION

Interactive data exploration (IDE), i.e., user-guided exploration of a database is an important task in data analysis. IDE tools such as Tableau assist users in exploring databases by allowing them to filter, group, and aggregate data through a visual user interface. The speed of computation when performing such operations is crucial. A study by Liu et. al. [23] showed that latencies greater than 500ms can have a significant negative impact on user performance. Thus, relational DBMSs are challenged to provide responses at interactive speeds for ad-hoc queries that users issue as they explore data through a visual user interface. However, traditional analytical DBMSs, such as MonetDB [25] or SAP HANA [14], often take seconds or even minutes to compute results on increasingly large databases. To mitigate this problem various query processing techniques have been proposed for DBMSs [2, 10, 11, 18, 22, 24, 34]. Examples include re-using previously computed results [9, 11, 13, 15], employing specialized data structures, on-demand creation of stratified samples [11], or performing speculative pre-computation based on previous interactions [20].

As the space of such techniques grows rapidly, comparing and evaluating their utility in the context of IDE becomes increasingly important. Yet, the workload used in traditional analytical SQL benchmarks such as TPC-H [38], TPC-DS [37], or the Star Schema Benchmark (SSB) [29] are not representative of the unique characteristics of IDE, and their metrics are not primarily intended to capture what matters to the end-user, rendering them unsuitable for IDE benchmarking. Our analysis of IDE user behavior shows that workloads are typically composed of sequences of related queries that are separated by time gaps. Users incrementally refine filters and visualize subsets of data using multiple plots for different attributes and aggregate functions. Conversely, queries found in existing benchmark are largely independent and unrelated, and meant to be executed back-to-back. Furthermore, metrics used in existing benchmarks are unable to fully capture aspects that are important to the end-user. They do

not take into account different execution models, such as *blocking* execution, where exact results are returned once all data is processed, and *approximate* and *progressive* models, which can be configured to balance the trade-off between result quality and query run-time. Moreover, the time a system takes to start up is largely neglected even though some engines heavily rely on pre-processing (e.g. to create samples offline).

We argue that an IDE benchmark must not be solely based on the end-to-end run-time of individual queries. Instead it should report on metrics that reflect that trade-off between speed and quality, as well as on the time it takes to pre-process data. Such metrics enable benchmark users and systems builders to answer questions like, "can a system provide responses within interactive latencies", "at what cost in terms of quality of the result", "at which data-size would a traditional column-store like MonetDB outperform an approximate engine?", "how much overhead do approximate query processing techniques introduce?", etc.

In this paper we present IDEBench, a new benchmark to facilitate comparison of database engines tested under common IDE conditions. The design of IDEBench is driven by the findings of five user studies we have conducted over the course of the past few years. First, contrary to existing benchmarks, IDEBench generates and uses time-spaced sequences of aggregate queries as its workload, which are representative of common query patterns we observed in the logs of our user studies. Our goal is not to accurately simulate users, which is arguably impossible, but instead to focus on a common denominator of IDE query patterns in query workloads that result from IDE systems. Second, IDEBench uses real-world data. This is crucial since reporting the quality of approximated results, i.e., accuracy and completeness, using synthetic uniformly distributed data, for instance, would be meaningless. To that end, we built a data generator that can be used to scale real-world datasets to any size while maintaining the original characteristics. Third, inspired by prior work of the data visualization community IDEBench reports on metrics that capture the trade-off between quality of results and query run-time. Finally, our benchmark is designed in an extensible way that enables users to add custom datasets and generate workflows that match their IDE use case.

In summary, the main contributions of this paper are: (1) We report on the results of an extensive workload analysis of five user studies. (2) We present the design of IDEBench, an extensible benchmark that facilitates the evaluation and comparison of different database engines for IDE workloads. The source-code of IDEBench is made available to the research community[1]. (3) We evaluated five different database engines using IDEBench and present and discuss their results: two commercial database engines, two research prototypes (IDEA, approXimateDB/XDB), as well as MonetDB.

[1]http://github.com/IDEBench/IDEBench-public

## 2 INTERACTIVE DATA EXPLORATION

In this section we first describe an IDE scenario that exemplifies how users visually explore data using IDE tools such as Tableau and Power BI. We then define the scope of our benchmark and present a selection of DBMSs that are within this scope.

### 2.1 An IDE Example

Jean, a research staff member at a major hospital, wants to get an overview of the hospital's patient population and their health problems. To do so, she looks at electronic health records from the past 20 years. Jean starts out by examining demographic information of patients and, for example, finds that patients ages are normally distributed. She then continues to look for interesting patterns in admission times and dates. Jean creates a query that shows the number of new admits per hour of the day. The result reveals that most admissions are during business hours, but there is an interesting bump from 7 to 10 pm. She filters down to admits from the emergency center and notices that most of the admissions between 7 and 10 pm were recorded there. Is this trend identical on all days of the week? She refines her query to display admits on weekends and sees that the bump shifted towards 10 to 12 pm. Who are these patients? Jean filters her previous age query by patients admitted on weekends between 10 and 12 pm. She finds that patients ranging from 20 to 35 are over-represented in this subset when compared to the overall age distribution. Now Jean wants to see which health problems are common among this sub-population. She finds that head traumas occur frequently and decides to check with the administration if the hospital's duty rota accommodates for this by making sure a trauma specialist is on call during weekend nights.

### 2.2 Scope of Benchmark

The above scenario illustrates a visual data exploration paradigm, which is commonly referred to as the *Visual Information Seeking Mantra*: "Overview first, zoom and filter, then details-on-demand" [32]. Modern IDE tools such as Tableau and Power BI are built to support this paradigm. On demand, users can create visualizations of attributes of interest, look at the distribution of associated values, and zoom into subsets of interest before inspecting specific instances.

Often visualizations can be turned into active filters; a technique known as *linking*. Most modern IDE tools can be configured to interpret selections of value ranges in a plot as filter that is subsequently applied to other plots presented

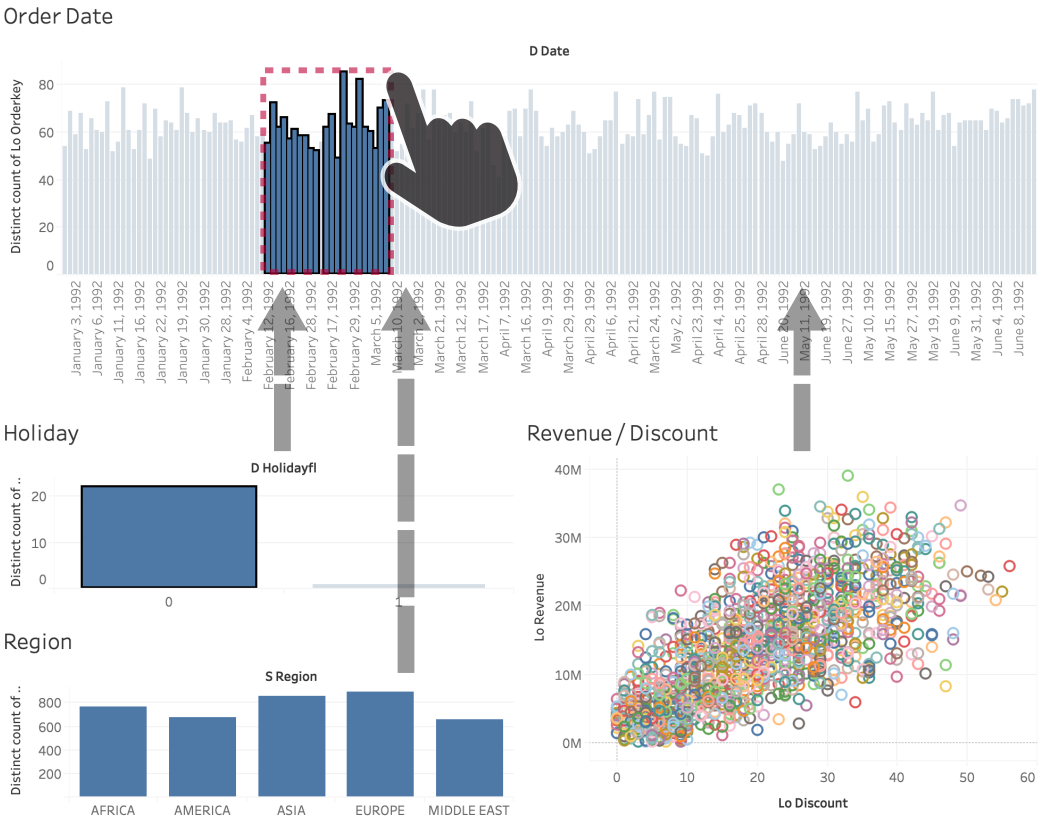



**Figure 1: A dashboard created by one of our user study participants using Tableau. She laid out multiple visualization in a coordinated view (the dashboard), where all visualizations are implicitly linked. In this example she selects a range of data points using a mouse gesture, filtering the three linked visualizations (shown by the grey dashed arrows). This selection results in three concurrent database queries.**

to the user. As a result, visualizations can be used to conveniently slice and dice data, and to examine sub-populations.

Unlike traditional dashboards, in which all plots are predefined and (implicitly) linked [24], modern IDE tools like Tableau and Power BI allow users to create custom dashboards, where plots can be arbitrarily added, removed, modified and linked. Interactions such as creating, modifying and linking plots, as well as selecting value ranges on an existing plot (filtering), trigger database queries. For instance, a user selection on a plot that is linked to $n$ other plots might results in $n$ concurrent queries. This paper focuses on IDE scenarios in which interactions like the ones outlined above are supported, and on structured (i.e., relational) datasets that are too large to be processed by a DBMS within interactive thresholds.

## 2.3 Database Landscape

Several commercial and academic DBMSs are either specifically built for IDE workloads, or can be used to support them. In the following, we summarize this landscape through three categories and provide examples for each.

*Analytical DBMSs.* This category represents database systems that efficiently execute SQL queries in an exact manner. It includes column-stores and main-memory systems such as MonetDB [25], SAP HANA [14], Hyper [21] as well as database management systems that are designed for online analytical processing (OLAP) type workloads [8]. Their execution model cannot guarantee interactive response times on large datasets.

*Approximate DBMSs.* Contrary to analytical DBMSs, approximate DBMSs use either offline or online sampling techniques to compute approximate answers. Most systems allow users to configure the trade-off between computation time

and the quality of the result through a time or quality constraint as part of a SQL query. Examples include AQUA [1], VerdictDB [39], BlinkDB [2], as well as approXimateDB [22].

*Specialized IDE DBMSs .* Prominent commercial examples of this category include Tableau [36] and its research predecessor Polaris [35], whose middle-layers are optimized for IDE workloads. ImMens [24] uses pre-computation over the entire query space to keep query run-times at a minimum. IDEA [15] is a middleware that provides interactive query execution on top of existing DBMSs such as Postgres or raw data sources such as CSV files. IDEA uses an online aggregation-based execution scheme to progressively compute and push query results to subscribers. Finally, DICE [18] is optimized for exploratory cube analysis and leverages interaction delays (i.e., "think-times") and a user interaction model to predict future queries.

## 3 THE NEED FOR AN IDE BENCHMARK

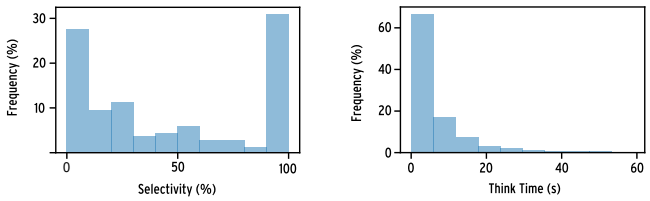
Although there are a number of existing analytical database benchmarks, this section highlights why they should not be used for IDE, why a new benchmark for IDE is needed, and which requirements it must meet.

### 3.1 Existing Analytical Benchmarks

Traditional analytical database benchmarks (TPC-H, TPC-DS, SSB) define a data-warehouse based workload with a fixed set of pre-defined SQL queries: TPC-H consists of 22 business oriented SQL queries, with a schema containing 8 tables (1 fact and 7 dimensions tables). TPC-DS comes with a much more complex schema containing 24 tables (7 fact and 17 dimensions tables) and 99 queries. SSB is based on TPC-H and re-organizes the database as a star schema with 5 tables (1 fact and 4 dimensions tables). SSB features 13 different queries, only some of which resemble TPC-H queries. Yet, as we explain in the next section, *none* of these benchmarks are representative of IDE workloads.

### 3.2 IDE Workload Analysis

To evaluate the applicability of existing analytical benchmarks, and to define benchmark requirements for IDE workloads, we draw on five independent user studies conducted with tools that are representative of what we outlined in Section 2.2. The five studies were conducted with 109 participants in total. Each study followed an open-ended (not tasked-based) protocol where participants were asked to explore real-world and synthetic datasets, and instructed to think-aloud and report insights that were noted down during the study. The only exception is user study 5, where we did not have the ability to create think-aloud protocols. However, we were able to use the interaction logs and query traces of the user study as basis of our analysis.




**(a) Selectivity** **(b) Think-Times**

**Figure 2: a) Shows the selectivity for all queries of user study participants. b) shows the distribution of think-times of all user study participants.**

*User Study 1 : Interactive Data Exploration Accelerators (IDEAs).* For this study 35 participants were recruited to explore a subset of the 1994 US census, using a visual IDE system called Vizdom that supports progressive visualizations using a pen-and-touch interface. Its frontend was designed to interactively explore data by arranging 1D or 2D aggregate plots of attributes on an unbounded canvas. Plots in the tool are interactive, meaning that users can link plots and select arbitrary bin ranges to create filter chains, and apply boolean operations (AND, OR, NOT) to combine multiple filters, etc.

*User Study 2: Tableau / Star-Schema Benchmark.* In this study, conducted as part of this paper, we invited 12 participants – all graduate students in computer science – to explore a dataset using Tableau. We used the SSB benchmark schema and generated a database with a scale factor of 0.1 ($\tilde{1}$00MB) to ensure that query results could be computed near-instantaneously. The goal of this study was to find out what whether the queries in SSB are representative of the traces that real users generate, and whether these traces overlap with the ones from study 1-3. Of the 12 participants, 8 stated to have some experience with data analysis, 4 reported to be advanced with data analysis (e.g., they have used Tableau, Power BI, or a similar visual exploration frontends multiple times). Each test subject was first given an overview of the database schema and a brief overview of the basic functionality of Tableau, e.g., how data can be plotted, filtered, and how multiple visualizations can be arranged in interactive dashboard views. The experimenter acted as a mediator between the participant and the software, i.e., the participants instructed the experimenter on what to do with the tool. We decided not to let the participants operate Tableau themselves as most pilot users struggled to remember the somewhat involved steps required to create plots, filters and dashboards.

*User Study 3: Effect of Progressive Visualizations.* The goal of this study was to investigate how progressive visualizations affect users during exploratory data analysis. The authors recruited 24 participants from a research university, all of whom had prior experience in data exploration or analysis tools (e.g., Excel, R, Pandas). The authors created a dashboard-like user interface with four visualization slots, that users can configure by assigning them any attribute from a dataset. All visualization are 1D/2D aggregate plots with configurable aggregate functions (count, min, max, avg). The four visualizations were implicitly linked; selecting values ranges in one visualization would filter/brush the other visualizations.

*User Study 4: Multiple Comparisons Problem (MCP) in Visual Analysis.* The aim of this work was to investigate the effect of MCP in visual analysis by evaluating the accuracy of user reported insights when exploring a dataset. This study required users to broadly explore datasets since they were explicitly asked to find interesting correlations in the data. All 28 participants were students who had some experience with data exploration or analysis tools (e.g., Tableau, Pandas, R) and have taken at least introductory college-level statistics and probability classes.

*User Study 5: DARPA Competitions.* As part of a larger DARPA program, we participated with our own data exploration tool in a competition where 10 government data analysts were asked to explore new datasets. We gathered and analyzed the query traces that resulted from the exploration sessions.

*Workload Analysis.* Given the results of user studies 1-5, we were interested to see (1) how users interact with IDE systems to make sense of data, how these interactions translate to SQL queries for a DBMS, and (2) if the workloads of existing benchmarks are representative of these queries. Our method involved analyzing multiple days worth of video and audio recordings, as well as the analysis of query logs from the underlying DBMSs. In the following, we discuss our main observations (O1 - O6).

**O1: User actions that trigger database queries** User actions that trigger database queries can be grouped into three abstract categories (independent of the IDE tool being used): 1) *creating/modifying visualizations*: the creation and modification of a plot involves settings parameters, such as the attributes that are being visualized, filters to apply to the data that is being visualized, as well as instructions on how to group/bin and aggregate the data. 2) *linking visualizations*: defining dependencies between plots so that they can be used as interactive filters 3) *selecting bins*: selecting a subset of the visualized data to filter linked plots.

**O2: Aggregation dominates.** Visualizing large datasets inevitably leads to over-plotting, which overwhelms users' perceptual and cognitive capacities. Aggregating data is the only practically reasonable way to support visual exploration of big data [24].

**O3: Selectivity varies significantly.** Queries formulated by study participants strongly varied in selectivity. We summarize the query selectivity of all queries from user study 3 in Figure 2, showing that approximately 30% of all queries either have a selectivity of $0 - 10\%$ while another 30% have a selectivity of $90 - 100\%$ respectively. Similar distributions emerged in user study 1.

**O4: Queries are built incrementally.** Users explore data and answer questions incrementally. Intermediate visualizations are used to inform further exploration steps. In all studies this was manifested by users either replacing parameters of an existing plot (e.g., the attribute being visualized), applying filters, etc. In user study 1, for instance, we noticed that the selections on plots were modified 6.91 times on average ($\sigma = 9$) to change a filter. In some cases users even changed the filter up to 50 times, trying to find interesting correlations for multiple minutes. In addition, we found that the number of attributes used to specify a query ranged from 1 to 5 different attributes (composed by AND and OR operators). We observed 5 different patterns of how the study participants used links to define dependencies between visualizations (see Figure 3):

- *Independent* (Figure 3a), where a user explores data by creating visualizations using individual queries and applying filters that only affect a single visualization at a time, as well as by altering the aggregation function (e.g., switching from `SUM` to `AVG`). This pattern was often used as a first step to browse through different attributes in a dataset.
- *1:N* (Figure 3b), where selecting bins one visualization triggers $N$ other queries for all dependent plots. This pattern, as well as the N:1 and N:N, typically apply after using *independent* visualizations to see how slicing and dicing the data affects previously plotted data.
- *N:1* (Figure 3c), where a user links $N$ visualizations to a single visualization. Selecting value ranges in any of the $N$ plots adds a filter predicate to the linked plot, forcing it to update.
- *N:N* (Figure 3d), where all visualizations are inter-linked. This pattern is typically found in interactive dashboards, where every visualization is implicitly linked with all other plots.
- *Sequential* (Figure 3e), is a browsing pattern where users create multiple visualizations that are sequentially linked. This pattern was often used when users drill down (zoom

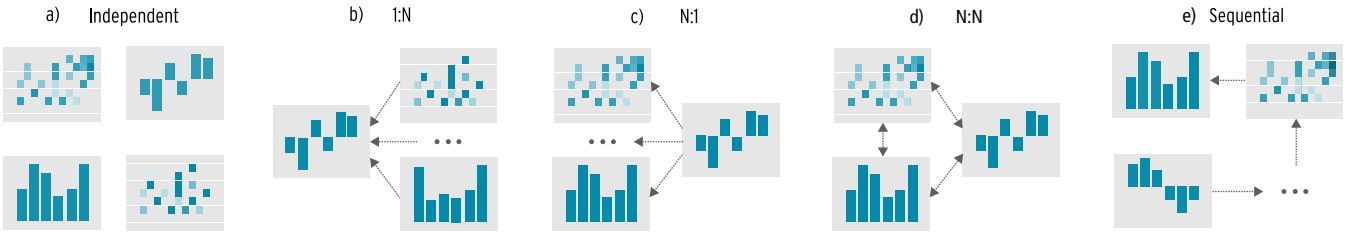



**Figure 3: Illustrates five different dependency patterns (see 4.1 for details) we observed during the IDE workload analysis of five user studies. a) Independent: four independent visualizations, changing the specification of any of the visualizations does not affect the others as there are no dependencies. b,c,d and e) an arrow represents a dependency from source to target. For instance, if a selection in the visualization on the left in b) changes, all N visualizations on the right need to be updated.**

in) into subsets of the data by building increasingly selective filter expressions, while keeping track of intermediate results.

**O5: Single interactions can lead to multiple concurrent queries.** Linked visualizations can lead to multiple concurrent queries. When users were free to arbitrarily create and link plots, they created dependency chains of up to 5 links (see Figure 3e, for example), meaning that a selection change on one plot would trigger up to 5 concurrent queries. Similarly, in user study 3, users were interacting with a predefined *N:N* dependency pattern between four visualizations in total. A selection in one plot led to an update in the other three visualizations.

**O6: Think-Time between interactions.**
We observed time gaps between user interactions (think-time) with high variance, ranging from just a few hundred milliseconds up to over 200 seconds in individual cases. Figure 2b shows the think-times recorded in user study 1 (outliers of more 60s were omitted). Most think-times were between 0 and 10s (mean=8.08s, $\sigma$ = 13.97), which could be leveraged by DBMSs to speculatively execute SQL queries to prepare for the next user interaction.

**O7: Real-data matters.** Users are less interested in actual values, but more in the distribution of values and/or irregularities/outliers that can only be found in real datasets. Similar observations were made by Amar et al. in a related study [3]. This became especially apparent in user study 2 and 4 where synthetic data was used. With synthetic data, users found it difficult to formulate meaningful queries as all attribute values were uniformly distributed.

Using these observations we derived a set of key requirements (R1 - R8) for an IDE benchmark. The list of requirements are an attempt to capture crucial aspects of measuring performance of a data engine in the context of interactive data exploration, and goes beyond what traditional benchmarks are designed for.

| | *TPC-H* | *TPC-DS* | *SSB* | *IDEBench* |
|---|---|---|---|---|
| *Schema* | snowflake | snowflake | star | star (default) |
| *Data Origin* | synthetic | synthetic | synthetic | real-world |
| *Data Distributions* | uniform | skewed | uniform | real-world |
| *Data Scaling* | yes | yes | yes | yes |
| *Iterative Query Formulation* | no | 4 out of 99 | no | yes |
| *Multi-Query Execution* | no | no | no | yes |
| *Think Time* | no | no | no | yes |
| *Metrics* | Time-based | Time-based | Time-based | Quality, Time |

**Table 1: A comparison of traditional analytical database benchmarks and IDEBench**

*Workload.* A benchmark for IDE should feature queries that resemble common IDE patterns. First, we note that based on O2 and O3, queries for aggregated plots have a common structure involving bins (group-bys), aggregated values (COUNT, AVG, etc.) as well as a set of filter predicates. For example, consider a dataset containing flight delays of domestic flights in the US [28]. A query for a plot showing the average flight delay per airport for flights originating in California can be structured as shown in Figure 4.

Second, it is pivotal that the benchmark not only runs individual queries in isolation (as done by TPC-H, TPC-DS, and SSB). Instead, as described in O4, the workload should contain time-spaced sequences of related queries mapping to the addition or modification of plots or selections therein

```
SELECT   AVG(DELAY) AS VALUE, AIRPORT AS BIN_AIRPORT,
FROM     FLIGHTS_DELAYS
WHERE    ORIGIN = 'CA'
GROUP BY BIN_AIRPORT
```

**Figure 4: An example query for a plot showing the average flight delay per airport for flights originating in California.**

(**R1**). Third, based on O5 the workload should representative of the fact that a single user interaction can lead to multiple concurrent queries (**R2**). Fourth, as stated in O6 and unlike TPC-H, TPC-DS, and SSB, a benchmark for IDE must honor that fact that there are think-times between user interactions, which data engines can leverage for speculative execution or other tasks (**R3**).

*Data.* Contrary to existing benchmarks, a benchmark for IDE should use a real-world dataset with outliers (O7). The benchmark must provide tools to scale the data while maintaining correlations and outliers (**R4**). Finally, as some data engines (e.g., BlinkDB [2] or IDEA [11]) only support single-table schemas, it is important that the data is available in both normalized and de-normalized form (**R5**).

*Metrics.* It is crucial that metrics go beyond measuring end-to-end runtimes of queries, and capture what matters to the end-user. Prior research suggests that there are two main factors that directly impact human performance in interactive data exploration: response time and accuracy. [5–7, 16, 17, 27, 30, 31]. For instance, Liu et. al. [23], showed that even response times of more than 500ms could lead to poor user performance. Researchers have also found evidence that poor accuracy or incomplete results (e.g., those of approximate and progressive systems) can lead to poor user performance [40] and false conclusions [12, 19]. Therefore, it is important that the metrics used in a benchmark for IDE reflect the trade-off between processing speed and the quality of the returned results: measures of responsiveness (**R6**), as well as measures of accuracy (**R7**).

*Customizablity.* Like traditional analytical database benchmarks, an IDE benchmark must define a default configuration for workload and dataset. However, as there is no one-size-fits-all solution for all parameters of a benchmark, we argue that it is important to provide the ability to customize the workload and data to match different IDE usage scenarios (**R8**). While some may argue that the ability to customize workloads, datasets and other parameters is undesirable as it hinders comparability, we, like [4] believe that customizability is crucial for adoption by different communities. Being able to configure such settings in a benchmark allows users to publish their configurations along with the benchmark results, and enable others to reason about the applicability, benefits and drawbacks of database designs and processing techniques in particular usage scenarios.

# 4 THE IDEBENCH DESIGN

To address the shortcomings of existing benchmarks we designed IDEBench, a collection of tools that can be used to evaluate the performance of databases on generated workloads that closely resemble the ones we observed in the logs of various different user studies (see Section 3.2).

## 4.1 Workflow Generator

Inspired by the observations we made when users visually explore data, we propose a customizable workload generator capable of generating series of common time-spaced user actions in IDE (*workflows*) which, directly or indirectly, trigger database queries. The workflow generator simulates user actions to create/modify and link visualizations (O1). The set of visualizations and actions used by the generator are abstract specifications that are independent of a concrete visualization tool.

*4.1.1 Visualization Specification (VizSpec).* A VizSpec is an abstract definition of a visualization, and is identified by an `id`. For instance, every visualization created in a Tableau can be thought of being specified by a VizSpec. It determines how data is grouped (`binning`) using one or many data `dimensions` and details how to perform numerical binning, e.g., using a `bin_width` or list of numerical `boundaries`. Using the `aggregates` attribute, an arbitrary number of aggregates per bin (e.g., the average of a numerical column and the total count of item in a bin) can be defined. The `selection` attribute specifies which data ranges have been selected on a visualization. The `depends-on` attribute specifies a dependency on selections of other visualizations. Finally, the time of execution of a VizSpec can be specified using the `time` attribute, (e.g., the time of a plot creation or modification). The time is specified in milliseconds relative to the start time of a workflow to simulate users' think-time.

*4.1.2 User Actions.* Based on observation O1 (Section 3.2), the workflow generator defines the following user actions.

- *Create/Modify:* An action to either *create a new* or *modify an existing* visualization. Each action is defined by a full VizSpec for creating a new visualization or a partial VizSpec for modifying a visualization (see Figure 5 and 8) respectively.
- *Link:* An action to link two existing visualizations, i.e. to establish a dependency from one visualization to another. It can be expressed by a partial VizSpec containing an `id` and one or more `ids` of visualizations it depends on (`depends-on`).

```
Visualization:
    id:          (string)
    time:        (number)
    binning:     (BinningDimension[])
    aggregates:  (Aggregate[])
    selection:   (string)
    depends-on:  (string)

BinningDimension:
    column:      (string)
    width:       (optional number)
    boundaries:  (optional number[])

Aggregate:
    type:        (enum COUNT|DCOUNT|AVG|SUM|MIN|MAX)
    dimension:   (string)
```

**Figure 5: A Visualization Specification**

- *Select:* An action to select a subset of the data in a visualization can be expressed by partial VizSpec containing its `id` and a set of selection predicates as a `selection` string.

*4.1.3 Query Model.* Any VizSpec $v$ can be mapped to a database query $Q(v)$ as shown in Figure 6 which is an abstract specification of SPJA queries. A query uses $v$'s bin (group-by) and aggregates specification, and its `filter` attribute is recursively computed as union of all selections of visualizations $v$ depends on. The `filter` attribute can be thought of as the `WHERE` clause of a SQL statement, comprising all relevant filter predicates. Joins between tables are implicitly defined in our query model by the attributes which are selected; i.e., we create equi-joins between those tables based on the given database schema of our data generator (see Section 4.2).

```
Query:
    binning:     (BinningDimension[])
    aggregates:  (Aggregate[])
    filter:      (string)
    time:        (number)
```

**Figure 6: IDEBench's Query Model**

Using such an abstract query specification, a concrete database driver of our benchmark renders an executable query. For example, if the database provides a SQL-like interface we generate queries as shown in Figure 8.

*4.1.4 Creating Workflows by Simulating User Actions.* Our workflow generator is designed to model interactive data exploration workflows, i.e. actions to create new and modify existing visualizations (**R1**). It uses as a Markov chain comprising three user actions: *create/modify*, *link*, and *select bins* (see Figure 7). We derived the transition probabilities between any pair of these actions empirically through the analysis of the study logs described in Section 3. The workload generator comes with pre-configured but customizable (**R8**) transition probabilities that can be used to generate variants of workflows mimicking the five dependency patterns shown in Figure 3 (an independent workflow, for example, has zero transition probability from create/modify to either link or select bins). To create a new workflow, the workflow generator samples $n$ actions from the Markov chain and adds a fixed inter-query pause (think-time) to consecutive actions by setting the `time` attribute of a VizSpec (**R3**) accordingly. We use fixed rather than dynamic think-times to be able to measure its effect on the accuracy of query results, as we shall see in the Section 6.4. An example workflow that was created by our generator for a 1:N workflow type is shown in Figure 8.

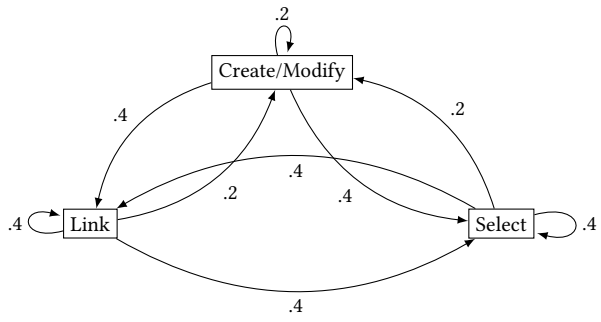



**Figure 7: Example of a Markov Chain used for 1:N workflows.**

Every user action exposes a set of parameters that are drawn randomly from a set of configurable probability distributions when creating a workflow (**R8**):

- *Create/Modify*: the number and types of binning dimensions (e.g., whether is a 1D or 2D plot), the bin width (i.e., the group-by granularity) or bin boundaries (i.e., minimum and maximum value used for each dimension), as well as the number and types of aggregates to use in the visualization.
- *Link*: linking strategy (none, 1:N, N:1, etc.).
- *Select Bins*: the bins to be selected as filter predicates.

Analogous to the transition probabilities between any of these actions, the defaults for the probability distributions of these parameters were informed by findings in the empirical analysis our user study logs.

When simulating an action the workflow generator recomputes the `filter` attribute for all affected queries and schedules affected queries for execution by setting the `time` attribute. To that end it keeps track of all VizSpecs or modifications thereof, by maintaining a directed dependency graph $G = (V, E)$. A directed edge in $G$ defines a dependency (link) between two visualizations, e.g., $v_1 \longrightarrow v_2$ indicates that $Q(v_1)$ must be re-executed when the specification of $v_2$, in particular its selection attribute is modified (**R2**).

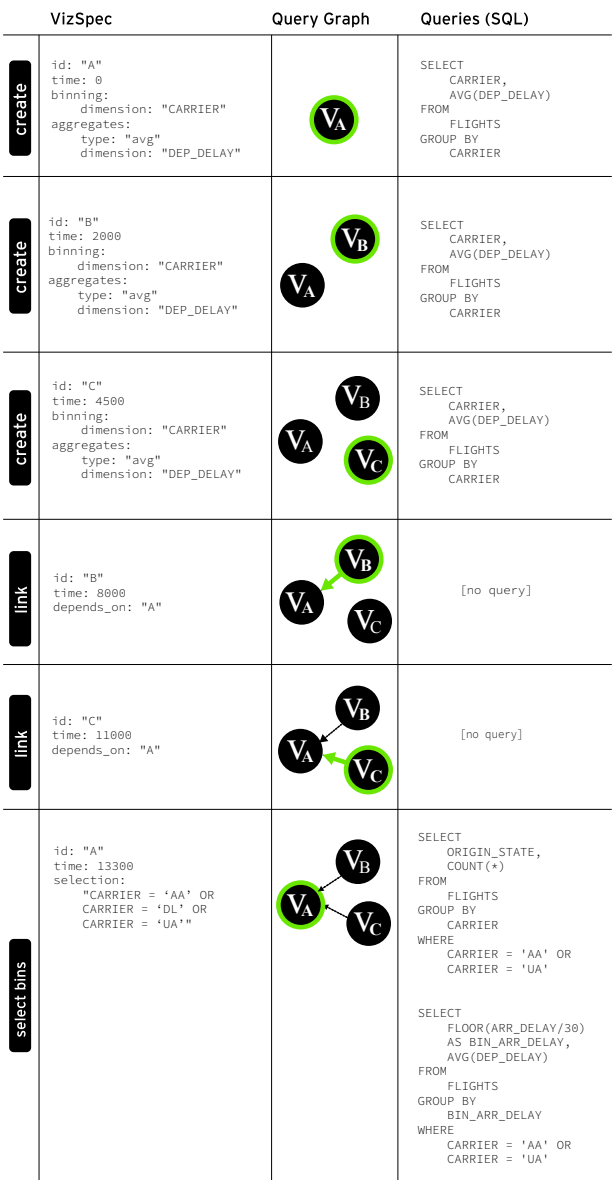



**Figure 8: An example of a generated 1:N workflow (Figure 3c.), the corresponding VizSpecs and translations to SQL. The highlighted item in the query graph corresponds to the current user action. Note that the final interaction triggers two SQL queries.**

## 4.2 Data Generator

Using real-world datasets in a benchmark for IDE is important as non-synthetic distributions make it harder for DBMSs to retrieve a representative sample, which consequently affects the quality of the results. Currently, IDEBench uses the U.S. domestic flight delay dataset [28] as default. However, users can customize the benchmark and plug in custom datasets.

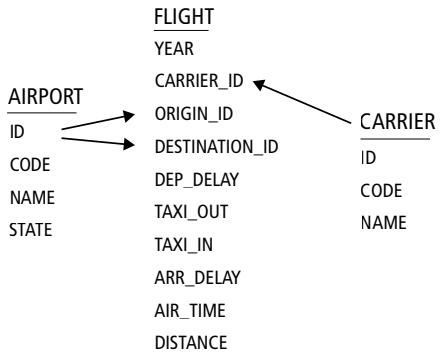



**Figure 9: The schema of IDEBench's default dataset.**

To scale a dataset to different sizes, IDEBench comes with a data generator that can resize a given dataset to any size (**R4**). The generator tries to maintain distributions in the data and relationships between attributes when scaling by simulating a Gaussian Copula [26]. The intuition behind this procedure is that every multivariate joint distribution can be expressed in terms of its marginal distributions and a function (the copula) which describes their relationship [33]. This decomposition allows for the creation (simulation) of new correlated samples.

In order to simplify the resizing process, IDEBench requires that the dataset is provided in a star schema-like format, in de-normalized form. This, however, is no limitation since typical IDE queries (i.e., SPJA queries) are natively supported on a star-schema. The algorithm to scale a dataset works as follows:

(1) First, we draw a random sample from the dataset.
(2) Second, we compute the covariance matrix $\Sigma$ and perform a Cholesky decomposition on $\Sigma = A^T A$.
(3) Finally, we resize the data. To create a new tuple, we generate a vector $X \sim \mathcal{N}(0, 1)$ of random normal variables and induce correlation by computing $\widetilde{X} = AX$. Furthermore, we transform $\widetilde{X}$ to a uniform distribution and finally use the CDF from the sample to transform the uniform variables to a correlated tuple.

Optionally, as a last step the data generator can split the generated data into multiple tables (fact and dimensions) based on the user-provided schema (**R5**).

## 4.3 Main Metrics

IDEBench computes metrics that capture the trade of between the runtime of a query and the quality/usefulness of the results.

*Data Preparation Time.* In IDEBench users are required to report on all actions taken to prepare for a benchmark run

(called "data preparation time" in our report). This includes the time it takes to copy the dataset into the system, to create sample tables/views offline, perform pre-processing, execute warm-up queries, etc.

Then, for every query, we compute the following metrics:

*Time Requirement Violations.* In IDEBench we measure the responsiveness of a DBMS using Time Requirements (TR). `TR Violated` is a boolean value indicating whether a query exceeded the time requirement specified in the settings (see Section 5.4). TR is violated if time TR after initiating the query no result is present or can be fetched. In practice, this means, that for batch-processing and AQP systems, TR is violated if the run-time of a query is greater than TR, and no intermediate result is present. For progressive systems, TR is violated if time TR after initiating a query no result can be fetched. Because IDEBench focuses on interactivity, it does not measure the actual query duration, i.e. it does not wait for query results to return if the computation takes longer than TR. If a query exceeds TR, database drivers can abort the query.

*Mean Relative Error (MRE).* Approximate and progressive results deviate from the ground-truth. To understand how much they deviate we measure the error between the *latest* result of an aggregate query and its ground-truth by computing the relative error, i.e., the ratio between the difference of the *latest* estimated result for a query $F_i$ and the actual result $A_i$ (**R7**).

$$\text{MRE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|F_i - A_i|}{|A_i|}$$

*Missing Bins/Groups.* Missing Bins/Groups is the percentage of all bins missing in the *latest* result for query. It is a measure of completeness for an aggregate query result, irrespective of the number of tuples processed by a system. The intuition is that the faster a system can enumerate all groups, the more insightful this will be to a user (**R7**).

$$\text{Missing Bins} = \frac{|bins\_missing|}{|bins\_in\_groundtruth|}$$

Missing bins are reported for exact and non-exact systems. While for exact systems the percentage of missing bins is either 0% or 100% (depending on whether or not a result was present in time), for non-exact systems we use the latest result that was returned within the given time-threshold. The mean relative error is only computed for results of non-exact systems.

*Cosine Distance.* Based on the observation that often users are more interested in characterizing data by its distribution rather than by the exact aggregated values [3], we additionally measure the cosine distance to test how much the "shape" of an aggregate result deviates from the shape of the ground-truth.

**Listing 1: A stub for a database adapter.**

```
class SampleAdapter:

    def workflow_start(self):
        # called before a benchmark run starts
        # do pre-processing, if applicable

    def run_query(self, viz_spec, result_queue):
        # 1. translate viz_spec into query
        # 2. execute query

    def workflow_end(self):
        # do clean-up, if applicable
```

# 5 THE IDEBENCH IMPLEMENTATION

IDEBench comprises two main components: a *benchmark driver*, which is responsible for simulating previously generated workflows, and *database adapters* which translate and delegate instructions from the benchmark driver to a database.

## 5.1 Benchmark Driver

The core of IDEBench is the benchmark driver, a command line application configured to load and simulate workflows created by the workflow generator. Given a workflow, the driver reads VizSpecs and builds and modifies an internal query graph structure, delegating queries and query updates to a database adapter (cf. Section 5.2). The database adapter then translates the VizSpecs from their JSON specification to the query language supported by the DBMS (e.g. SQL).

## 5.2 Database Adapters

In order to benchmark an IDE system using IDEBench, a *database adapter* must be implemented, which acts as a proxy between the benchmark and the system that is being tested. The benchmark driver delegates changes made to the query graph to a database adapter, which then translates them into queries supported by the database (e.g., SQL). There are three core methods that must be implemented by a database adapter: `workflow_start`, which can be used to run pre-processing tasks before a workflow starts, `run_query` to initiate new or update existing VizSpecs (e.g., when adding a filter, see Listing 1), and `workflow_end` which is called once the workflow has ended, allowing the database adapter to perform clean-up tasks.

| Setting | Description | Default |
|---|---|---|
| Workflows | The workflows to run in the benchmark | 10 |
| Time Requirement (TR) | The maximum execution duration for a query | 0.5s |
| Dataset and Size | The dataset to run the benchmark on and the number of rows in the fact table | Flights 500M |
| Think Time | The delay between two consecutive interactions | 3s |
| Using Joins | Whether a star schema is used where dimension tables are pre-joined to fact table | False |
| Confidence Level | The confidence level at which an AQP/progressive systems return confidence intervals | 95% |

**Table 2: Benchmark Settings**

## 5.3 Benchmark Defaults

By default IDEBench uses 10 generated workflows using a mix of the four query graph patterns described in Section 4.1. Table 2 shows a summary of the most important parameters of IDEBench and their defaults, which we defined based on findings in our user studies. It uses the flight delay dataset (introduced in Section 4.2) scaled to 500M tuples in de-normalized form (i.e., only one large fact table). It also sets the time requirement TR for each query to 0.5 seconds, uses a think-time of 3 seconds between each user interaction, and a confidence level of 95%.

Upon completion of a benchmark run, IDEBench generates two reports: (1) An aggregated summary report listing how frequently the time requirement was violated, how many bins are missing on average, and the distribution of mean relative errors for all queries which did not violate the time requirement. Figure 10 shows an example of such a summary report. (2) A detailed report listing all settings and metrics on a per-query basis.

## 5.4 Customizing the Benchmark

For the sake of comparability, we encourage users of IDEBench to use the default configuration. However, all parameters can be modified so that users of IDEBench can test their database systems with settings that match a specific target use-case. For instance, some DBMS might be specifically optimized for data exhibiting certain distributions, for very low latencies, or for a workload that only supports specific types of queries.

# 6 AN EXPERIMENTAL STUDY

To demonstrate the applicability of IDEBench and show the effects on the different classes of query processing engines (2), we conducted an experimental study using following DBMSs:

(1) *MonetDB:* a state-of-the-art open-source analytical DBMS, which uses a blocking query execution model that requires users to wait until an exact query result is computed. Thus, upon initiating a query, the run-time of a query is unknown. (2) *approXimateDB/XDB:* a PostgreSQL-based DBMS that supports online aggregation using the wander join algorithm [22]. It allows for a maximum run-time to be set in SQL. In addition, a "report interval" can be set so that intermediate results can be retrieved at fixed time intervals. XDB has some limitations in terms of query support, which we describe in detail in Section 6.2. (3) *IDEA:* a system that supports online aggregation and uses a fully progressive computation model; after initiating a query, results can be polled at any point in time. (4) *System X:* a commercial in-memory AQP engine operating on stratified sample tables that are created offline. The run-time of queries cannot be specified explicitly but must be indirectly adjusted by varying the size of samples tables. (5) *System Y:* a commercial specialized SQL engine for IDE that uses a blocking execution model, which is designed as an in-memory optimization layer on top of various different DBMSs.

In the remainder of this section, we describe the configuration and setup of our experiments, and discuss the results and findings.

## 6.1 Configuration and Setup

*Configuration.* We used the default flight delay dataset (see Section 4.2), with S=100 million, M=500 million (default), and L=1 billion tuples in de-normalized form (i.e., a single pre-joined table). Furthermore, we used five different time requirements (TR) 0.5s, 1s, 3s (default), 5s, and 10s, and a fixed confidence level of 95% (default). While various Human-Computer-Interaction studies recommend time requirements of less than 1*s* (see Section 3), we also included larger ones to get a better sense of how fast results converge. Because none of the systems use speculative query execution in their default configuration, we set the think-time to a fixed value of 1*s*; we analyze the effect of varying the think-times in a separate experiment (see Section 6.4).

*Setup.* We ran IDEBench on *MonetDB*, *approXimateDB*, *IDEA* and *System X*, but were unable to run the full benchmark on *System Y* due to the absence of an API that could be used in a database adapter. However, we executed selected queries of our benchmark manually through its user interface in a separate experiment (see Section 6.5). All experiments were conducted on a computer with two Intel E5-2660 CPUs (2.2GHz, 10 cores, 25MB cache) and 256GB RAM. We used the default parameters for all DBMSs without hand-tuning them and did not tweak or optimize any of the system parameters.

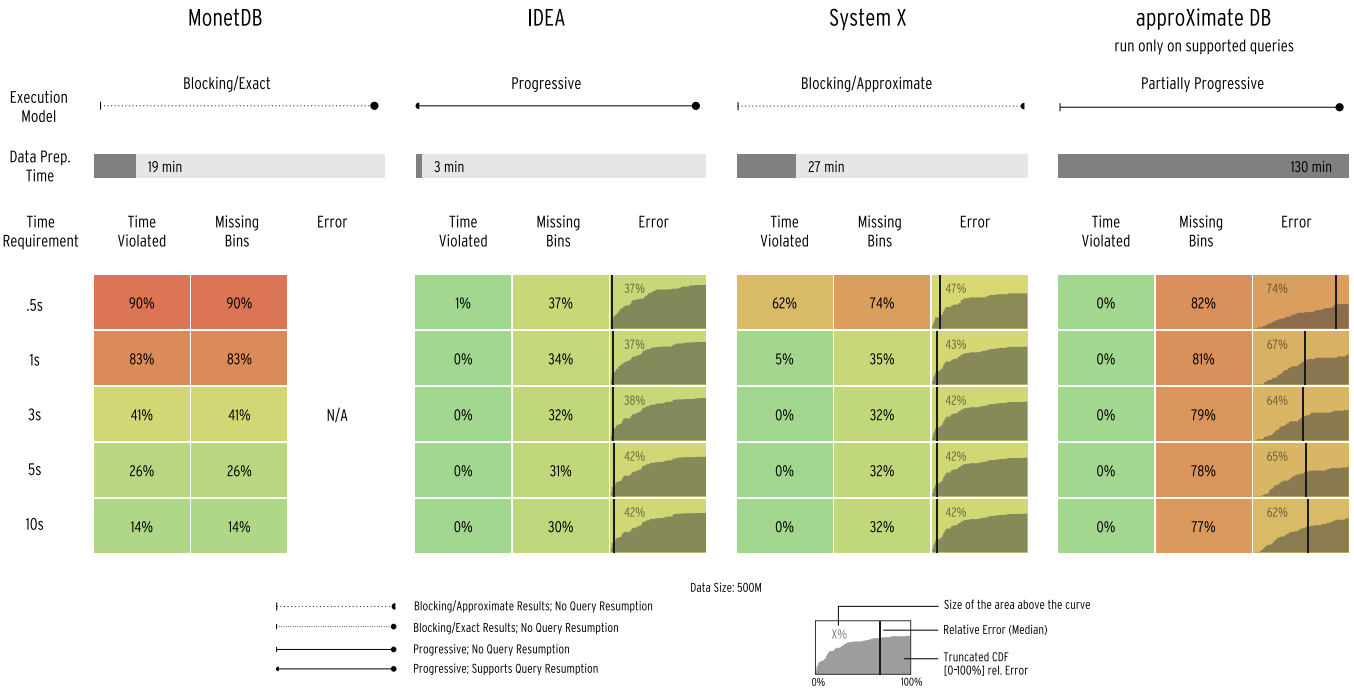



**Figure 10: Shows the aggregated benchmark results for four systems in a summary report. The benchmark was run for five time requirements on a dataset with 500M rows. It shows the mean percentage of time violations and missing bins, as well as a CDF of the mean relative errors (MREs) truncated to errors less or equal to 100%. Thus, the greater the proportion of small errors, the smaller the area above the curve (shown as percentage above the CDF).**

## 6.2 Exp. 1: Overall Results

In our main experiment (Figure 10 and 11) we analyzed how the four systems behave with respect to different time requirements (see Figure 10). We show the results for 10 workflows of the default configuration, a data size of 500M tuples, de-normalized schema, and a confidence level of 95%.

*Data Pre-Processing.* In *MonetDB* data stored in a CSV file can be loaded into the database through an SQL interface, which takes approximately 19 minutes for 500M records. *IDEA* expects data in a single randomized CSV file. On start-up the system loads a configurable number of tuples into main memory, which took approximately 3min in our experiment. With *System X* data stored in a CSV file can be loaded into the database through a SQL interface. In order to be able to execute approximate queries, stratified sample tables have to be created offline. We used a sample size of 1% of the dataset size. *System X* further requires that each connection executes a warm-up query when the system is restarted. For 500M records, we measured a data preparation time of 27min. Finally, *approXimateDB* took 130min to load and prepare the data. While this system provides support to pre-load relations and indexes into the database buffer in main memory, we did not make use of this feature for our experiments.

*Speed and Quality Metrics.* As expected for an exact execution model, *MonetDB*'s TR violations decrease roughly linearly with the time requirement, and so does the percentage of missing bins (see also Figure 11a). On the contrary, *approXimateDB* never violates the TR as the query run-time can be specified accordingly. However, it is important to note that this system only supports online aggregation for `COUNT` and `SUM`, but does not support `AVG` and multiple aggregates in a single query. Therefore, we excluded all unsupported queries (34% of all queries) for *approXimateDB*. For comparison, running the experiment for all queries (falling back to regular Postgres for unsupported queries) leads to TR violations of 66% for all TRs, i.e., all non-approximate queries exceeded TR.

With *System X* more than 60% of all queries violate TR=0.5s. Interestingly, for TR=1s only 5% are violated, and for TR=3s all query results are returned on time. The percentage where TR is violated is therefore a good indicator of how large a sample table needs to be, if speed is more important than result quality. *IDEA* does not violate any TR, with the exception of 1% of all queries for TR=0.5. The authors confirmed

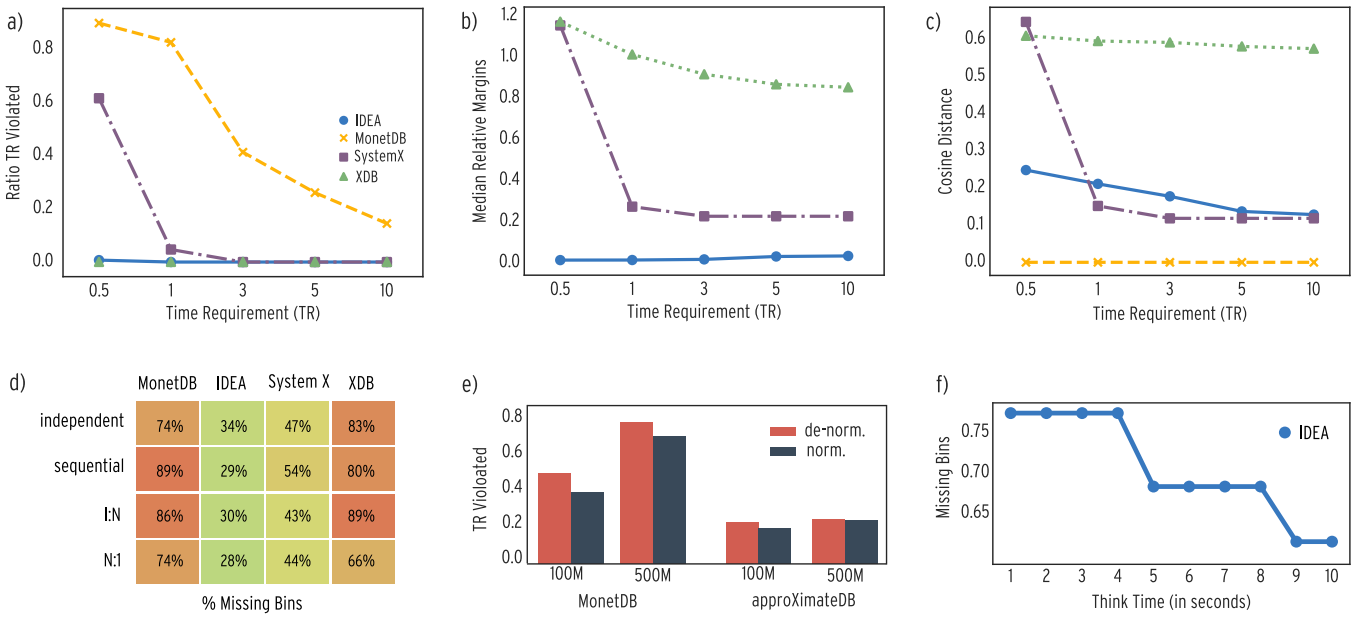



**Figure 11: a, b, and c) show how the ratio of TR violations, the median of the mean relative margins, and the cosine distance behave for increasing time requirements. d) Compares how the percentage of missing bins differs depending on which system and workflow type is used. e) A comparison of the percentage of violated time requirements for MonetDB and *approXimateDB*, using a normalized and de-normalized dataset of size 500M. f) Shows the effect of varying think-times on missing bins for *IDEA*.**

that this is due a slightly higher overhead for the first query after a restart of the system. *IDEA* also starts off with significantly less missing bins (37%) for TR=0.5 than any other system, but achieves similar performance as *System X* for TR>=1s. Furthermore, *IDEA* manages to perform better than other systems in terms of mean relative error of all returned results. The median of all mean relative errors is significantly less than *approXimateDB*'s and *System X*.

Another interesting fact can be observed by inspecting the cumulative distribution of mean relative errors. In Figure 10 (column *Error*), *approXimateDB*'s "area above the curve" is significantly greater than the one of *IDEA* and *System X*, indicating that high mean relative errors occur more frequently. A similar conclusion can be drawn by looking at the end of the curve. For instance, *approXimateDB*'s curve ends, below 50%, indicating that more than 50% of all mean relative errors are greater than 100%.

Figure 11b and 11c show how the median of the mean relative margins, and the cosine distance changes for the different DBMSs with increasing time requirements. Again, *approXimateDB* has significantly greater relative margins than both *IDEA* and *System X*. Moreover, while *System X*'s median is close to 120% for TR=0.5s and drops to approximately 20% for TR=1s, *IDEA*'s median remains constant around zero for all TRs. Finally, Figure 11d compares how the proportion of missing bins differs based on the system and query graph patterns. As none of the systems we used in the evaluation use speculative execution by default, there are only few notable differences per column. For instance, *MonetDB* has fewer missing bins on average for "independent browsing" and N:1 patterns, which may be attributed to the fact that interactions of these workflows only trigger a single query.

## 6.3 Exp. 2: Varying Schema Complexity

In this experiment, we compare the performance of *MonetDB* and *approXimateDB* using a normalized and de-normalized schema. Contrary to experiment 1, we set up *approXimateDB* so that any query that cannot be executed online will fall back to a regular Postgres query. We exclude *IDEA* and *System X* as they do not support joins. Using the data generator we created two datasets of 100M and 500M tuples and normalized the data so that the fact table holds foreign keys to two dimension tables (airports and carriers). Interestingly, as can be seen in Figure 11e, both *MonetDB* and *approXimateDB* perform slightly better in terms of time requirement violations with a normalized schema since the overall scan volume of data significantly decreases. *MonetDB*'s proportion of TR violations grows with the size of the normalized

dataset, while *approXimateDB*'s TR violations remain steady due its ability to perform joins online.

### 6.4 Exp. 3: Varying Think-Time

In this experiment, we evaluated the impact of increasing think times between interactions (see Figure 11f). We used an experimental extension of *IDEA* that speculatively executes queries when two visualizations are linked. For this experiment, we used a fixed data size of 500M tuples, a time requirement of 3 seconds, and created a custom workflow comprising two linked visualizations. Internally, *IDEA* uses a simple speculative execution procedure. While user think IDEA starts queries for every possible single bin selection in the source visualization. If too many bins exist, it selects only the $k$ visually most dominant ones. If upon the next interaction one of the bins is selected, *IDEA* can return a potentially better estimate of the results, as the query was given more processing time. Figure 11f shows the results of this experiment with the proportion of missing bins for ten different think times (1s - 10s).

### 6.5 Exp. 4: Experiment with System Y

In the last experiment, we manually executed a selected subset of our workflows in a commercial IDE System Y and used MonetDB as a backend. We used a fixed data size of 500M and performed three variants of the 1:N workflow type. In particular, we were interested to see whether *System X*'s middle layer that pre-fetches and pre-computes query results leads to significant benefits over using a vanilla MonetDB setup. However, we did not find this to be the case. System Y renders and updates the visualizations in the workload roughly at the same speed as if *MonetDB* is used directly.

### 6.6 Discussion

Our experiments with IDEBench and five systems have shown that the performance on IDE workloads in terms of data preparation time, responsiveness as well as the quality of the results can vary significantly, and thus these systems are not equally suitable for IDE.

*Preparation Time.* Creating representatives samples offline is challenging: users need to find a suitable sample size to balance the trade-off between processing speed and quality of the results. Although the time overhead of offline-sampling approaches can be reduced by employing online sampling instead, systems using online sampling such as *approXimateDB* may still require much time to load data into the DBMS. *IDEA*, for instance, bypasses the loading problem by reading samples from disk at runtime. The system, however, requires the data to be randomized prior to ingestion.

*Time Violations.* We found that progressive and AQP systems like *IDEA* and *System X* were able to keep time violations at a minimum while maintaining low error rates with increasing data sizes and time requirements. This is in stark contrast to traditional analytical databases represented by *MonetDB* where time violations increase for larger datasets and time requirements. We also found that *approXimateDB* can only execute a subset of the queries in our workload online. It has to revert to executing a significant number of queries in a blocking fashion, which leads to notably more TR violations. Finally, given the log of queries that a user executes during an exploration workflow, systems can leverage think-times to speculatively execute queries in order to provide faster responses upon the next user interaction.

*Error Metrics.* Progressive systems progressively refine results and therefore lower the errors and confidence intervals over time, converging to an exact result. AQP systems that create sample tables offline, on the other hand, have constants error rates and confidence intervals, irrespective of the time requirement. To optimize the error rates for such systems, users must find representative sample that is small enough not to violate the time requirement, which can be challenging. Finally, we observed weak completeness scores, i.e. high missing-bin values for low time requirements and in workflows containing concurrent queries (such as sequential or 1:N).

## 7 CONCLUSION

In this paper, we presented IDEBench, a new benchmark designed to evaluate systems for interactive data exploration (IDE). Unlike traditional analytical database benchmarks, IDEBench's workloads and metrics are inspired by the requirements and usage patterns of real IDE systems that were derived through a number of different user studies. Using our benchmark we conducted an evaluation that included five different DBMSs (approximateDB, IDEA, MonetDB as well as two commercial systems) representing three different system categories (traditional exact DBMSs, approximate/progressive DBMSs, as well as specialized engines for IDE). The results showed that especially for low latency requirements approximate and progressive query processing engines outperform traditional databases.

## 8 ACKNOWLEDGEMENTS

This research is funded by the DARPA Award 16-43-D3M-FP040, NSF Award IIS-1562657 and NSF Award IIS-1514491 and supported by Google, Intel, and Microsoft as part of the MIT Data Systems and AI Lab (DSAIL).

# REFERENCES


[1] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *ACM SIGMOD*, pages 574–576, 1999.

[2] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 29–42. ACM, 2013.

[3] R. Amar, J. Eagan, and J. Stasko. Low-level components of analytic activity in information visualization. In *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, pages 111–117. IEEE, 2005.

[4] L. Battle, R. Chang, J. Heer, and M. Stonebraker. Position statement: The case for a visualization performance benchmark. *IEEE Internet Computing*, 13(3):48–55, 2009.

[5] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool. The effects of loss and latency on user performance in unreal tournament 2003®. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 144–151. ACM, 2004.

[6] J. Brutlag. Speed matters for google web search. https://services.google.com/fh/files/blogs/google_delayexp.pdf, 2009.

[7] S. K. Card, G. G. Robertson, and J. D. Mackinlay. The information visualizer, an information workspace. In *ACM SIGCHI*, pages 181–186. ACM, 1991.

[8] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1):65–74, 1997.

[9] A. Crotty, A. Galakatos, K. Dursun, T. Kraska, C. Binnig, U. Çetintemel, and S. Zdonik. An architecture for compiling udf-centric workflows. *PVLDB*, 8(12):1466–1477, 2015.

[10] A. Crotty, A. Galakatos, E. Zgraggen, C. Binnig, and T. Kraska. Vizdom: interactive analytics through pen and touch. *PVLDB*, 8:2024–2027, 2015.

[11] A. Crotty, A. Galakatos, E. Zgraggen, C. Binnig, and T. Kraska. The case for interactive data exploration accelerators (IDEAs). In *HILDA@SIGMOD*, page 11. ACM, 2016.

[12] G. Cumming and S. Finch. Inference by eye: confidence intervals and how to read pictures of data. *American Psychologist*, 60(2):170, 2005.

[13] M. El-Hindi, Z. Zhao, C. Binnig, and T. Kraska. Vistrees: fast indexes for interactive data exploration. In *ACM SIGMOD*, page 5, 2016.

[14] F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, and J. Dees. The SAP HANA database – an architecture overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012.

[15] A. Galakatos, A. Crotty, E. Zgraggen, C. Binnig, and T. Kraska. Revisiting reuse for approximate query processing. *PVLDB*, 10(10):1142–1153, 2017.

[16] P. Hanrahan. Analytic database technologies for a new kind of user: the data enthusiast. In *ACM SIGMOD*, pages 577–578. ACM, 2012.

[17] J. Heer and B. Shneiderman. Interactive dynamics for visual analysis. *Queue*, 10:30, 2012.

[18] P. Jayachandran, K. Tunga, N. Kamat, and A. Nandi. Combining user interaction, speculative query execution and sampling in the dice system. *PVLDB*, 7:1697–1700, 2014.

[19] S. Joslyn and J. LeClerc. Decisions with uncertainty: the glass half full. *Current Directions in Psychological Science*, 22(4):308–315, 2013.

[20] N. Kamat, P. Jayachandran, K. Tunga, and A. Nandi. Distributed and interactive cube exploration. In *ICDE*, pages 472–483. IEEE, 2014.

[21] A. Kemper and T. Neumann. Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *ICDE*, pages 195–206. IEEE, 2011.

[22] F. Li, B. Wu, K. Yi, and Z. Zhao. Wander join: Online aggregation via random walks. In *ACM SIGMOD*, pages 615–629. ACM, 2016.

[23] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE transactions on visualization and computer graphics*, 20:2122–2131, 2014.

[24] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. In *Computer Graphics Forum*, volume 32, pages 421–430. Wiley Online Library, 2013.

[25] Monetdb. http://www.monetdb.org. Accessed: 2019-11-02.

[26] R. B. Nelsen. *An introduction to copulas*. Springer Science & Business Media, 2007.

[27] J. Nielsen. Powers of 10: Time scales in user experience. *Retrieved January*, 5:2015, 2009.

[28] B. of Transportation Statistics. Bureau of transportation statistics. http://www.transtats.bts.gov, 2017. Accessed: 2019-10-21.

[29] P. E. O'Neil, E. J. O'Neil, and X. Chen. The star schema benchmark (ssb). *Pat*, 200(0):50, 2007.

[30] S. C. Seow. *Designing and engineering time: The psychology of time perception in software*. Addison-Wesley Professional, 2008.

[31] B. Shneiderman. Response time and display rate in human performance with computers. *ACM Computing Surveys (CSUR)*, 16(3):265–285, 1984.

[32] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343. IEEE, 1996.

[33] M. Sklar. Fonctions de repartition an dimensions et leurs marges. *Publ. inst. statist. univ. Paris*, 8:229–231, 1959.

[34] Snappy data. https://www.snappydata.io/. Accessed: 2019-11-02.

[35] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Trans. Vis. Comput. Graph.*, 8(1):52–65, 2002.

[36] Tableau. http://www.tableau.com. Accessed: 2019-11-02.

[37] TPC-DS. http://www.tpc.org/tpcds/, 2016. Accessed: 2019-11-02.

[38] TPC-H. http://www.tpc.org/tpch/, 2016. Accessed: 2019-11-02.

[39] VerdictDB. Verdictdb. https://www.verdictdb.com. Accessed: 2018-05-30.

[40] E. Zgraggen, A. Galakatos, A. Crotty, J.-D. Fekete, and T. Kraska. How progressive visualizations affect exploratory analysis. *IEEE transactions on visualization and computer graphics*, 23(8):1977–1987, 2017.