UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering,
Mathematics & Computer Science

QuestionMark: designing a benchmark for probabilistic databases

Nikki Zandbergen

M.Sc. Thesis
July 2023

Supervisors:
dr. ir. M. van Keulen
dr. T. van Dijk

Advisor:
ing. J. Flokstra

Data Management & Biometrics
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Preface

This master thesis is written to obtain the degree of Master of Science Computer Science at the University of Twente. The subject of data science and database technology has interested me since my time in secondary school. After my bachelor in Business and IT, I wanted to dive deeper into the topic of data science, and thus my choice of master specialisation was made. In the course *probabilistic programming*, the topic of probabilistic databases was introduced to me. I found the technology intriguing and it made me want to research it further. I was engaged in writing this thesis from April 2022 until June 2023.

I hope this thesis inspires its reader to step outside of the known boundaries and explore the possibilities that probabilistic data processing has to offer. It opened my eyes and I am sure it will open yours too.

I want to thank my supervisors for supporting me during this long time. Maurice van Keulen, Tom van Dijk and Jan Flokstra, thank you for supporting me also in times where working on my thesis did not go as smooth as I had hoped for. Maurice, your enthusiasm on this theme inspired me into working on this topic.

I also want to thank my family for supporting me throughout. You were always a listening ear for me and you provided me with all the love and support I needed.

In loving memory of grandma.

No cats were harmed in the making of this thesis.

Nikki Zandbergen
Enschede, July 2023

Abstract

As increasing volumes of uncertain data are produced every day, the need for a mature probabilistic database management system grows. Various probabilistic database systems have been developed throughout the years, but none seems robust enough to function in a real-world environment. To aid the development of a robust system, The QuestionMark Benchmark for Probabilistic Databases has been developed. QuestionMark is a benchmark specifically designed for real-world strain testing of probabilistic databases. QuestionMark covers a wide range of functionalities, so that any application area can be tested. To validate the performance of the benchmark, the state-of-the-art probabilistic database MayBMS and the novel probabilistic database DuBio are run through the benchmark to evaluate their effectiveness, efficiency and appeal. Empirical evaluation shows that QuestionMark is a promising technology and can fulfil its purpose.

Contents

1

Introduction

Although the field of probabilistic databases has been studied for over two decades, a breakthrough in its use outside the academic world has yet to happen. The theoretical foundations of probabilistic databases have been established and the first few prototypes have shown their potential. However, the corporate world remains loyal to the deterministic processing of data, even while a large part of the data produced today is incomplete or uncertain.

Having uncertain data treated in a deterministic manner ignores the many opportunities that treating that data in an indeterministic manner offers, and it might even lead to incorrect decisions due to incorrect data displaying [20]. Probabilistic data processing can aid decisions in more scientific areas, such as bio-informatics [92] and healthcare [62], but also adds value in various business processes, which rely on decisions based on data from different sources. To get uncertain data ready for deterministic decision-making, data cleaning is performed to remove inconsistencies in the data. This process consumes significant time, while the risk of making wrong decisions due to badly cleaned data is still present. Probabilistic data querying solves this issue. Being able to query raw business data in a probabilistic manner provides an improved information representation to base business intelligence decisions on [20]. It is thus not the case that the availability of good quality probabilistic databases only aids the scientific world; on the contrary. A wide range of sectors could benefit from the use of probabilistic DataBase Management Systems (DBMS).

To enable this step towards real-world usability, a standardised manner of verifying the performance and functionality of probabilistic databases should be established. Benchmarking can deliver this [54]. Benchmarking provides a fair comparison of two different systems. While many benchmarks are available for deterministic databases, such as TPC and SPEC [54], there is no current standard for probabilistic databases. Being able to benchmark probabilistic databases can play a crucial role in establishing widespread use of probabilistic databases in the real world.

As data is often treated as if it were certain, it might not even be obvious to those who manage the data that what they process is actually uncertain. Uncertain or incomplete data is common in real-world scenarios and can be retrieved from many areas.

These include sensor data [3, 13, 44, 50, 74], scientific data collection [7, 48, 70, 94], data integration (data deduplication) [11, 23, 87, 49, 63, 70, 76], user profiling [94], medical data [5, 48, 59] and human-entered data [7]. Data can become uncertain due to measurement errors, noise, incompleteness, or inconsistencies [3], though it could also be that the data is deterministic, but our understanding of that data is uncertain [23]. Being able to express and query this uncertainty in a non-deterministic manner is important for a thorough understanding of the data and to enable well-established decisions based on that data [70].

Traditional databases are designed to store exact data and are limited in their ways to handle uncertain data. This makes it difficult or sometimes impossible to work with uncertainty in data, causing many application opportunities to remain untouched [44, 74]. To unlock these opportunities, probabilistic databases have been developed. Probabilistic databases can omit the cost of enforcing certainty in data and can enable applications that were otherwise unexplored [23]. Unfortunately, no probabilistic database management system to date performs well enough to be used in various real-world scenarios [91].

To add to the research on probabilistic databases and to join the movement towards real-world usability, a benchmark will be designed. The goal of this benchmark is to deliver a standard for testing probabilistic database systems for real-world usability. When this benchmark is designed, the novel probabilistic database DuBio [88] and the state-of-the-art probabilistic database MayBMS [6] will be run on the system to verify the usability.

During this research there will be a special interest in product matching. Product matching is a form of entity resolution [49], which refers to the process of identifying which data entities from multiple sources refer to the same real-world entity [65, 96]. Entity resolution aids the goal of data integration, which should lead to an increased data quality and size usable for further analysis [96].

The task of product matching is not trivial. When combining product data from more than one source, chances are that two or more sources disagree in the value of an attribute of a single product [65]. In that case, it should be determined what source contains the correct attribute value, which is impossible to do both reliably and efficient. Another issue is determining what products are the same in the first place, as conflicting attributes in product offers make the same products appear different. To solve issues like these, probabilistic data integration can be used. With probabilistic data integration, products from disagreeing sources can still be used. Their product information will then be displayed with an uncertainty about the possible values.

Although the product matching case will be used to evaluate the performance and showcase the possibilities of a probabilistic DBMS, the use of this research is not limited to product matching. Various business processes rely on the integration of systems with overlapping information or on the processing of inherently uncertain data. Having a system that can translate this uncertainty and display it to the user of the system can enhance business intelligence decisions and tackle the issues discussed earlier.

To support the entry of probabilistic databases in business applications, this paper aims to answer the following research questions:

RQ1 How can a benchmark be designed to test and compare probabilistic database management systems on real-world strain?

RQ2 How do the novel probabilistic database DuBio and the state-of-the-art MayBMS perform when benchmarking these technologies with the developed benchmark?

The rest of this paper is structured as follows. In Section 2 an overview will be given on previous work on probabilistic databases, benchmarking, product matching and the technologies used in this research. Section 3 provides the theoretical framework on which the benchmark design will be based and Section 4 discusses the methodology used to create the dataset, design the benchmark and verify the performance of this benchmark. The full benchmark design will be presented in Section 5. The benchmark will then be put to the test with DuBio and MayBMS, and the results are discussed in Section 6. The conclusion of this research is presented in Section 7. The discussion, future work and limitations are also discussed in this final section.

The manual provided with QuestionMark is provided in Appendix A. In Appendix B more information on the used dataset is provided. Appendix C contains an evaluation on the performance of QuestionMark: The Dataset Generator. Additional information on the performed user study can be found in Appendix D. The raw results from the empirical evaluation can be found in Appendix E.

2
Background

In this chapter, the definition of a probabilistic database will be given and a background on probabilistic databases will be provided. Previous work on benchmarking technologies for databases will be discussed and the shortcomings of those will be evaluated. Product matching of datasets will also be discussed. Finally, the technologies used in this research are presented and discussed.

2.1. Probabilistic Databases

Although probabilistic databases could be seen as an extension to traditional deterministic databases, the data they process is vastly different. Where databases were traditionally designed to only include deterministic data, data generated nowadays is increasingly more uncertain. Because of this, probabilistic databases were developed.

In general, a probabilistic database models a set of possible databases, as opposed to a single one in a traditional database [11]. Probabilistic databases are systems that store uncertain data and support complex queries that translate this uncertainty to the user [22]. In probabilistic databases, uncertain data is annotated with a confidence score. This confidence score is interpreted as a probability and thus mathematical computations can be performed on them [22]. By having these uncertain attributes, different possible databases can be constructed. The set of possible databases is also referred to as the set of possible worlds, where each database instance is a representation of a possible world. When additional evidence is provided to the dataset, it could be that certain possible worlds are not true anymore. These worlds are then removed from the set of possible worlds and the probabilities of the remaining worlds are normalised [47].

Theoretically, when a set of possible worlds would be queried, the query answer would be the average of the result that the query would return in each possible world separately [90]. In reality, implementations of probabilistic databases are more complex. If all possible worlds were to be modeled and an exact probability calculation over all of those would be performed, execution time would be exponential [76].

Although the exact implementation of different probabilistic databases vary, they are all developed to serve the same goal. Earlier research has identified various properties that a probabilistic DBMS should possess. These properties include scalability, expressiveness, succinctness, efficiency, genericity and convenience [9, 48, 90].

Within this research, the following definitions are used regarding probabilistic databases:

- *Possible world:* A possible world is an element from a set of possible worlds, where $p^{[i]}$ is its probability and R_k denotes the amount of relations in that possible world [46].

$$\langle R_1^i, \ldots, R_k^i, p^{[i]} \rangle \in W$$

- *Probabilistic database:* A probabilistic database is a finite set of structures, where each set of relations within the structure has a valid probability [46].

$$W = \{\langle R_1^1, \ldots, R_k^1, p^{[1]} \rangle, \ldots, \langle R_1^n, \ldots, R_k^n, p^{[n]} \rangle\}$$
$$where \sum_{1 \leq i \leq n} p^{[i]} = 1.$$

- *Descriptive sentence:* A descriptive sentence describes a subset of possible worlds using the notation of set theory [89]. This research uses the term *sentence* for conciseness.

The idea of having uncertain databases has been researched for long. First researches on uncertain databases date back to the early 1980's [34, 42], where the focus lay on managing databases containing null-values and large dependencies, and providing a mathematical foundation for these theories. In 1997, the first prototype of an uncertain database system was released, called ProbView [50]. Today, the focus has shifted towards practical implementations of databases that can manage uncertain and incomplete data conveniently.

It was found early that probabilities are a very powerful, yet complex tool for managing uncertainty [50]. Although earlier research identified several ways in which uncertain data could be modelled [23], the focus of the research community eventually shifted to a probabilistic approach for the implementation of uncertain databases. Hence why they are called probabilistic databases in this research. Different probabilistic databases have been developed, attempting to provide a system that can be used in the real world, but with no success. Even to date, no probabilistic database system is able to provide near-exact probability calculations on larger amounts of data [76, 92].

Over the years, some of the more promising probabilistic databases developed are MayBMS [7], MCDB (Monte Carlo Database) [43], and Trio [94]. Although these all serve the same goal, the internal functioning of each of these systems differ. On a high level, MayBMS and Trio provide probabilities based on tuple-level uncertainty, while MCDB calculates these based on attribute-level uncertainty [86]. The expressive power of these uncertainty models also differ. When assigning probabilities to tuples, these probabilities are independent of each other, as with Trio. MayBMS solved this independency issue by providing world set descriptors, which describe this uncertainty relation [55].

Recently, a new type of probabilistic database has emerged: open world probabilistic databases. In normal probabilistic databases, it is assumed that the true world exists among one of the possible worlds. In an open world, the possibility is added that the true state could be one not present among the known possible worlds [19, 18]. As the challenges of managing open world probabilistic databases have not yet been solved, focus in this research is merely on closed world probabilistic databases.

2.2. Benchmarking

In order to properly and fairly test a new piece of software, benchmark testing can be used. A benchmark provides a standardised manner to test the performance and functionality of a specific type of software. It can aid in making meaningful comparisons between different technologies and identify opportunities to improve the tested software. For a benchmark to be fair, it should work indiscriminately [46]. This implies that the benchmark itself may not favour one technology over the other and should focus on the environment instead of the system.

From the Oxford English Dictionary, *to benchmark* has the following definition:

- *Benchmark:* To evaluate or check (something) by comparison with an established standard; to measure against a comparable or equivalent point of reference, esp. in order to assess performance or set performance standards [61].

Benchmarking was originally defined as a process of understanding what is of importance for the success of a business. Benchmarking in this context is a continuous process that produces hard data, where best practices from other companies are used to perform quality improvement in the business [51, 56]. This definition is tweaked when benchmarking was used also in areas like software engineering, where focus now lies on evaluating the performance of a specific software against one or multiple reference systems [69].

The process of benchmarking software technologies is an essential step towards real-world usability, as it brings several advantages that would be hard to obtain otherwise. When benchmarking a technology, its performance is compared to that of similar technologies. The benchmark can reveal potential areas of improvement and can aid in the establishment of that improvement by showing how competing technologies deal with the area [56]. This learning is best done by a benchmark that provides both qualitative and quantitative measures. A good benchmarks provides an incentive for improvement and delivers the tools to do so [56]. A benchmark additionally provides convenience in testing technologies. Without a benchmark, a technical specialist should be hired to test a system thoroughly, which would cost a business many resources in terms of time and money [59]. Apart from these benefits for the developers of the software technology, the benchmark also provides a standardised way to compare various technologies for potential consumers [54].

Designing a proper benchmark is no trivial task. A benchmark for database technologies consists of both a dataset and a set of queries. Both the dataset and queries need to be representative of the real-world use that the technology will encounter. When these are both established, the design of testing should be considered.

Numerous benchmark tools are already available for testing deterministic database management systems. These include Apache JMeter, BAPco, SPEC, TPC and Wisconsin [33, 54], of which TPC is the most well-known. The deterministic relational DBMS PostgreSQL also provides its own benchmark with its software. With this, users of the system can perform their own benchmark tests [83].

For probabilistic databases, no standard benchmark tools are available yet [85]. Lately, more research is being conducted to provide this. LUBM is a benchmark that can be used and scaled to handle an arbitrary number of probabilistic statements in the context of SparQL [73]. MayBMS created an adaptation of the TPC-H benchmark to test their system, called Probabilistic TPC-H [48]. Other researches also designed their own benchmark to test MayBMS, such as [15] and [85]. However, none have reached a widespread use within the research community.

One of the reasons these benchmarking technologies do not suffice to be used in the real world, is that they do not make use of a real-world undeterministic dataset. Probabilistic TPC-H and the two MayBMS benchmarks make use of synthetic data and synthetic uncertainty. Other benchmarking technologies focus more on uncertainty in other types of uncertain data, such as Linked Open Data with LUBM.

Currently, there is thus no benchmark tool that can be used as a standard for testing probabilistic database management systems. There is a need to bring the real-world application of these technologies towards a novel benchmark.

2.3. Product Matching

The benchmark presented in this research stimulates real-world load of probabilistic database management systems in the context of entity resolution and product matching. Entity resolution is the process of identifying what data entities represent the same real-world entity [11, 57, 65]. It is the first step in the process of data integration and product matching [57]. Product matching is also called schema matching or ontology matching and is a process that can be applied to heterogeneous sources of structured or semi-structured data that describe the same real-world entities [10, 96].

Identifying similar products in different sources is a challenging task, as the data on a single product is almost always inconsistent. One source might provide richer data than the others, or a source might include a completely different attribute value [2, 72]. Different product titles may describe the same entity, but the same title might also describe two different entities [10]. Data can also differ due to different naming conventions, the use of abbreviations, typographic errors, missing values or data obsolescence [11, 49, 63]. A website might also choose to display a product differently due to marketing decisions [65]. The lack of a labelling standard for products on e-shops also complicates the process of product matching [72].

The area of product matching is widely studied and commonly applied in various corporate settings. For instance, there are numerous comparison websites that collect product data from different web shops. These provide the user of the platform with an overview of all product providers and help them select the best possible deal [2]. The continuous growth of the e-commerce industry makes that ever more and diverse product information becomes available. With this growth, the interest in product matching increases and more powerful product matching technologies are needed to process these volumes of data [2, 49].

Ayat *et al.* [11] motivated that while research on entity resolution for certain data has been well studied, research on applying entity resolution on uncertain data has still been lacking. Moreover, research on entity resolution mainly focuses on providing the user a deterministic answer. This raises issues with incorrect product matches, essentially showing the user wrong information. In this research, the goal of product matching is to present the user with a confidence score in the match, once this match is sufficiently uncertain. By doing so, the user will be notified by the possible incorrect match, allowing them to be critical of the shown results themselves. Undoubtedly, if time were of no concern, humans would outperform any product matching software on the market today.

In that sense, the area of product matching matches the philosophy of probabilistic databases well. As cited by [20], the importance of doubt has been stressed since ancient times:

> *"If a man will begin with certainties; he shall end in doubts; but if he will be content to begin with doubts, he shall end in certainties." (Francis Bacon, 1605)*

> *"Doubt is one of the names of intelligence." (Jorge Luis Borges, 1889)*

The area of entity resolution and product matching allows for a good platform to showcase the possibilities of probabilistic databases and can aid in demonstrating the added value that using probabilistic databases can bring.

2.4. Technologies

For this research, two probabilistic relational database management systems are used and put to the test with the designed benchmark. Both are built upon PostgreSQL, a deterministic DBMS. In this section, each of these systems will be introduced, and their identified strengths and weaknesses from earlier research will be discussed.

2.4.1. PostgreSQL

PostgreSQL [82] is an open source relational database management system. Its design was first presented by the University of California in 1986 and was originally called POSTGRES [78]. Now, PostgreSQL is developed and maintained by the PostgreSQL Global Development Group and has grown to the fourth most used relational database management system worldwide [77].

PostgreSQL has earned strong reputation by providing a rich relational database management system, able to run on all major operating systems, while being free and open source. It offers a reliable high-performance system and is fully ACID-compliant [82]. Additionally, PostgreSQL is fit as a solid foundation for any extension project. Since it is built with extendibility in mind, features such as custom data types and functions can easily be added. It also makes use of the liberty license, meaning that its source code can be freely adapted and distributed for any purpose [83]. Hence why many projects, including MayBMS and DuBio, run on top of PostgreSQL.

2.4.2. MayBMS

MayBMS [6] is one of the first relational database management systems able to manage uncertain and incomplete data. MayBMS was first introduced in 2006 and promises a space and time efficient query execution with scalable evaluation [7]. Their aim is to have a robust database system that could be used in real applications [46]. MayBMS is developed on top of PostgreSQL in a way to ensure a fully integrated system [7, 46]. The current version of MayBMS requires to be run with PostgreSQL 8.3.3, released in 2008, and offers all functionality present in that version [48].

The internal functioning of MayBMS is based on possible worlds theory [7] and uses U-relational databases [9]. U-relations focus on record-level uncertainty. For this, three additional columns are added to a record per uncertainty: one displaying the random variable, one for its value, and the third containing the probability [46]. Because of this design, MayBMS supports complex dependencies [55].

Regarding the query design, MayBMS uses an SQL-like language to query the probabilistic data. It hides the complexity for the user and rewrites and optimises the queries once submitted. The query language is an extension on the SQL syntax, with a few adjustments. MayBMS dropped the support for standard SQL aggregates and introduced new probabilistic aggregates and constructs for dealing with incompleteness and probabilities. [8].

Experiments conducted by the developers of MayBMS showed that the system can fulfil the expectations of being a usable probabilistic DBMS. These experiments showed a runtime execution close to that of conventional query evaluation [7] and suggest that MayBMS will perform well in real-world scenarios [5, 48].

Various researches investigating the usefulness and opportunities of probabilistic databases have used MayBMS as a prototype. [3] and [59] used MayBMS for the purpose of detecting faulty sensors and modelling patient counts respectively. Both researches found MayBMS to be of great use. Also [92] reported MayBMS to function well for the given bio-informatics task.

Although MayBMS was very competitive for its time, it also contains properties that make its use less practical for real-world applications. The following issues were reported in earlier research on MayBMS:

1. There is a limit to the amount of random variable assignments MayBMS supports due to the design decision of adding three new columns per random variable [76, 92]. This issue is due to a restriction from PostgreSQL on the maximum amount of columns per table [82], which will be reached when constructing large and complex sentences. A maximum of 500 random variable assignments can be supported per record [92].
2. MayBMS cannot handle OR-relations in sentences [7]. When having to digest

an OR-relation, it will be translated to an AND-relation with negations. This complicates the formulated sentences, requiring more columns to store these sentences.

3. A query run with *aconf()* will always return an approximation of the probabilities, even if returning an exact probability is possible or even more time-efficient [76].
4. The performance of MayBMS is not always stable. [15] noted that MayBMS reported errors and memory issues on certain runs on random data, especially when data sets grew over 1 million records. [76] reported a rapid growth of runtime when facing larger datasets.
5. For the purpose of the research of [76], MayBMS did not work out-of-the-box and had to be tweaked to support relations of arbitrary arity and provide a fair timing.

2.4.3. DuBio

DuBio [88] is a novel probabilistic database developed by the University of Twente. Although not yet officially introduced, its source code can already be downloaded to test the system and run experiments. The aim of DuBio is to provide real-world scalability of probabilistic data processing on complex queries. DuBio is built as an extension of PostgreSQL [20], making it less dependable on a specific PostgreSQL version and allowing it to run on several versions, including the current latest version PostgreSQL 14.3.

The internal functioning of DuBio is also based on the possible worlds theory [91], and focusses on record-level uncertainty. DuBio represents this uncertainty using a type of Binary Decision Diagrams [20]. DuBio tries to address issues found in other probabilistic database management systems to deliver a system usable for real-world applications [91]. One of these improvements is that DuBio uses a single column to store sentences, allowing the storage of significantly larger sentences than would be possible with MayBMS, while still providing a compact representation. This is done by holding a dictionary of variables, which is a complex structure stored in a separate database table in a single cell [20]. This also allows for native expression of OR-relations.

The query language used by DuBio is an extension to SQL with added constructs for dealing with probabilistic data. Unlike MayBMS, DuBio has not yet released a query language that hides the complexity of the processing to the user. The current method of querying is explicit and only meant to be a temporary solution. Work is being done to provide a simpler query language for DuBio [37].

As DuBio is a new system, no external research has been conducted on its usability and performance. Internal research has identified DuBio to be a promising technology [20, 71], although development is still in progress to improve the time efficiency of complex queries even further.

3
Theoretical Framework

In this chapter, the theoretical framework of database benchmarking will be provided. Providing an overview on the foundations of benchmarking and database benchmarking specifically aids in substantiating the design decisions made in this research. The benchmark in this research will follow best practices from benchmarking generally and from earlier research on and with database benchmarking. Fairness of database benchmarking will also be discussed.

3.1. Benchmark Foundations

As discussed in Chapter 2, benchmarking is used to properly and fairly test a piece of software. For a benchmark to have these properties, it needs to be well-designed. That said, it is crucial to know what 'well-designed' means and how this could be established.

The basics of benchmarking as we know it now have been formed around 1970, where an increased interest in benchmarking emerged in the research community [80]. Although there is still no consensus on a universal benchmarking process, the flow of benchmark testing discussed in the various sources is mostly similar. The differences are partly due to the wide range of systems and processes that can be tested with benchmarking, each requiring a tailored approach and focus. The benchmark designed in this research is no different.

A benchmark, in general, measures three aspects. In order of decreasing importance, these are:

- *Effectiveness,* which relates to the quality of fulfilling the purpose;
- *Efficiency,* which relates to the use of resources and to execution speed;
- *Appeal,* which relates to the human element, including satisfaction of use.

Some studies that use benchmark testing to showcase the abilities of the tested system focus mainly on the efficiency and omit effectiveness and appeal in their benchmark test. Examples can be found in [15] and [54]. A good benchmark captures all these three aspects and guides the user in interpreting the results so that meaningful conclusions can be drawn on each aspect.

The effectiveness, efficiency and appeal of a system can be measured using different metrics. A metric measures various business processes or software specifications and expresses these in numbers. Common metrics express aspects as cost, risk, throughput or total execution time. Since a benchmark test should be kept simple, decisions need to be made on what metrics are relevant for the system or process that is being benchmarked. Often a combination is used of generic, easy to obtain metrics and system or process specific metrics [35]. The metrics for a benchmark should also be carefully considered. It is important to include both qualitative and quantitative metrics that can be expressed with sufficient precision, and to offer the context in which these metrics matter and how the results should be interpreted [56]. When the metrics are specified, it should be explained how these metrics can be obtained.

The processes that produce these metrics consume data doing so. When benchmarking business processes, the process itself as well as the data that is fed into that process are often both evaluated when performing the benchmark study. When benchmarking software systems, it is common to have a dataset delivered with the benchmark as to ensure reproducibility when testing different pieces of software. For the benchmark to be meaningful, this dataset should be a representation of the data that the software or process digests during its real-world use.

When the overall benchmark design it laid down, details of the benchmark design should be established. These details include the amount of iterations required per metric and the estimated duration of the benchmarking exercise.

When benchmarking, it is important to iterate the tests. More iterations lead to more reliable results. By iterating, the case that an underlying unknown variability influences the results is minimized [33]. Earlier benchmark researches varied between three and ten iterations [29, 33, 48, 85, 96]. The average over these iterations is then used as the value of the metric. When measuring the performance of simple queries, hardware performance could influence the execution time more than the duration of the query itself. In these cases, the query can repeatedly be run and the total execution time is measured over these runs [15, 43].

It can also be a good indication for the user to show the duration of the benchmarking exercise. Providing this can help businesses in setting up targets and deadlines [24].

Apart from these benchmarking elements, earlier research has also identified several criteria that a well-designed benchmark should comply to. The following seven characteristics of a well-designed benchmark were identified by [35], [40] and [56]:

- *Fairness.* The tested systems should be compared fairly;
- *Interoperability.* The benchmark should be easy to implement on different systems and has a common format;
- *Relevance.* The benchmark performs operations typical for the domain;
- *Representative.* The benchmark performance metrics are accepted by industry and academia;

- *Reproducibility*. The benchmark is versioned. Earlier benchmark tests can be reproduced and verified;
- *Scalability*. The benchmark should be scalable to larger and smaller systems and should work on parallel computer systems;
- *Simplicity*. The benchmark should be transparent and understandable.

These seven characteristics are widely used to verify the validity of a benchmark. Other characteristics, such as 'ethical practices' mentioned in [51], can also be used. However, as these characteristics were found to be of lesser relevance to the benchmark designed in this research, elaboration on these is omitted.

When the benchmark is designed, it should be implemented in the business to gain intelligence from the benchmark test. Most benchmarking processes described in literature have a strong business focus. The benchmark processes described in literature vary in the amount of steps and phases and in the type of benchmarking applied. In general, benchmarking contains the following phases [4, 31, 51, 56]:

1. *Planning*. The benchmarking subject and partners are identified. Data collection methods are also determined in this phase;
2. *Analysis*. The benchmarking information is gathered. The current competitive gap is determined and an analysis is made on where the company wants to be;
3. *Execution*. The benchmark is used to measure performance and is recalibrated where needed;
4. *Learning*. The benchmark results are analysed and changes are implemented where required.

A benchmark thus has three basic building blocks, which are the process to be measured, the metrics and the dataset. All elements should be carefully considered when designing the benchmark. To guide the user of the benchmark, details of the benchmark and how it should be implemented should also be specified. Details include the amount of iterations required and the estimated time required to perform the benchmark test. A well-designed benchmark also covers the effectiveness, efficiency and appeal of a process or software, and complies to the seven characteristics mentioned in this chapter.

3.2. Database Benchmarking

When it comes to testing database technologies, the potential of having a standardised benchmark test was noted. The first database benchmark tests were published around 1990 [35] and research in the area has continued ever since. Database benchmarking is now a common manner of testing new and existing database technologies. An array of database benchmarks have been published, where each benchmark targets a different type of database technology or a different use case of the technology. With all the research on database benchmarking, new design practices have emerged.

In a business context, benchmarking is usually used to compare a processes to one similar within the own company or a rivalry business. When benchmarking software systems, a company often wants to know what technology provides the best performance to cost payoff for supporting their current business processes. It is therefore of importance that the benchmark can be tweaked to imitate these real-world business process as closely as possible.

One important part of database benchmarking is the benchmark dataset. Benchmark datasets are datasets designed for use in database benchmarking and have specific properties that distinguish them from ordinary datasets. They should contain large volumes of data, reference data should be available, and they should be free to access [16]. However, as there are many different specific application domains, it could be challenging to find a benchmark dataset fit for the purpose of the technology to be benchmarked. One alternative to finding a real-world dataset would be to synthetically generate data. Although synthetic data could never capture the diversity of real data, it does bring advantages. Synthetically generated data is always available, gives full experimental control and never infringes copyrighted material or privacy sensitive information [29]. A combination of both could also be used, where additional synthetic data can be generated from the real-world dataset. In this way, additional data could be created to increase the dataset size, or a full new privacy respecting dataset can be generated that follows the patterns in the original dataset.

Apart from the data itself, there are other properties of a dataset that should be considered when designing a benchmark. To allow for scalability of the benchmark, it should be possible to scale the dataset of the benchmark [26]. To also ensure reproducibility when scaling the dataset, it should be indicated how the dataset must be scaled when running experiments. Other factors that should be adaptable by the user are whether to allow for data partitioning, indexing, redundancy and data reorganisation [26].

In database benchmarking, queries to test the system should also be provided. For this, it is important to know the selected dataset throughout. Only then, the queries of the benchmark can be formulated. These queries should provide full coverage of the different types of operations supported by the system and should be a realistic representation of the real-world environment the system will be deployed in [26]. These operations should include selections, projections, joins, sorting and aggregation. The queries should vary between specific queries, targeting a part of the database, and broad queries, targeting the database as a whole.

When designing the set of queries, the accompanying metrics should also be considered. As with the queries, the metrics should also provide a complete picture of the system. Choosing the wrong metrics can give a wrong impression of the system, making the benchmark ineffective [26]. Choosing a wide range of metrics is thus of importance. As one system might perform well on a specific task, but poor on another, it would provide a skewed view of its performance if only one task was measured [35]. Common metrics in database benchmarking are load time, size of the dataset, query response time, throughput rates, prize of computing hardware and power consumption. Normalized metrics include price per throughput and power per throughput [26].

When benchmarking database management systems, new areas of concern arise. For example, a database technology may or may not comply to the ACID properties and the benchmark may or may not cover this as being an important aspect. A benchmark could also choose to assume such properties are present and provide the dataset and queries with that in mind.

Database benchmarking is in many aspects similar to other areas of benchmarking, but it does require its own focus. With database benchmarking there is a special interest for the benchmarking dataset and the accompanying queries and metrics. As database benchmarking often does not take place in a set business environment, additional care is required to ensure that the benchmark is as true to the business as possible. Other aspects, such as the importance of ACID compliance, should also be included.

3.3. Fairness

When designing the benchmark, one thing that should be kept in mind is fairness. When comparing two different database technologies with the same benchmark, both technologies should be treated in the way that is optimal and fit for their purpose. Although it is not the main task of a benchmark to stress this importance, it is best to indicate the ways in which the benchmark can be altered to fit the technology that is being put to the test.

The importance of fairness has been mentioned in several research papers [21, 26, 40, 46], where the papers of Hohenstein and Jergler [41] and Raasveldt *et al.* [68] analysed this aspect of benchmarking specifically. Both conducted their study on the fairness of comparing technologies using benchmark tests and found that promoters of new technologies often predicate that their systems outperform current standards. However, the tests performed to get those results were often not fair and favoured the novel technology. To not get trapped in the same pitfalls, special attention will be paid to making this benchmark and the tests that will be performed with it as fair as possible. In [41] and [68], the following aspects are presented that must be avoided in a performance comparisons:

1. *Scope of comparison.* You should always compare one specific system to another. Comparing one new technology to the class of older technologies creates invalid conclusions;
2. *Using a small test data set.* When running tests with a small dataset, the execution time is mostly dependent on the in-memory capabilities of a system, instead of the technology;
3. *Test with a warm start.* Most real-world applications use a wide range of data, making that the cache is continuously replaced with different data. Running queries on the same small set of data multiple times improves the execution time and provides a wrong impression of the tested technology;
4. *Using standard configurations.* A DBMS has many configurations that can be tuned to optimize the performance of a system. When benchmarking the traditional database technologies, often standard configurations are used. This can heavily influence the performance of the tested technologies;
5. *Over-tuning the system.* Configurations of a system could also be used to optimize the system specifically for the few scenarios that the benchmark covers.

When using such a system in a real-world scenario the promised execution times will then never be reached;

6. *Using artificial test scenarios.* Artificial test scenarios are abstractions of the reality and often use a configurable number of nodes and relationships. Although this makes testing easier, they do not provide a proper representation of their real-world strain. Additionally, these tests rarely consider concurrent execution of various query types as is often the case in real-world applications;
7. *Not using a system's full potential.* In most database management systems, features are available to improve the performance of the system, such as stored procedures. Ignoring these features and only using standard SQL does not provide a realistic image of the performance;
8. *Data distribution.* When dealing with big data, the data distribution might impact the performance. This happens especially when a different start node is chosen by each system to evaluate the query;
9. *Perform biased evaluation.* As a benchmark test comprises of a combination of different results, these need to be aggregated to get a global picture of the performance. By interpreting the results in the right way, a system with an average performance could have a seemingly better performance;
10. *Non-reproducibility.* The possibility to reproduce experiments is a key element of scientific verification. When the benchmark experiments cannot be reproduced, the results cannot be verified and thus cannot be trusted;
11. *Incorrect code.* When testing a new system, bugs in the code might positively influence the performance of the system. When the produced results are not properly checked, it might be that the bug caused the query to skip parts of the data, resulting in a faster execution time.

When designing a benchmark, the importance of the above points should be emphasized and compliance with them should be included in the design as much as possible. Knowing that a benchmark is designed with fairness in mind creates trust in the produced results.

4 Methodology

In this chapter, the methodology used in this research is provided. The methods are discussed for designing the benchmark, selecting the dataset, performing probabilistic product matching and executing the benchmark.

4.1. Designing the Benchmark

For the main part of this research, the design of the benchmark is established. The benchmark has been designed following the principles of design science. Within this design process, special attention was put towards incorporating the benchmark principles presented in Section 3. The final benchmark design is presented in Section 5.

From the book of [95] and the paper of [66], the following definition of design science is provided in the context of this paper:

- *Design science:* The design and investigation of artifacts to serve human purpose in the context of the research field.

In the case of this research, the context is the use of probabilistic databases with real-world product matching data. Although design science was initially not developed for use with information system design, the general acceptance and the research towards a fitting framework has increased [36]. This results in a framework fit to solve problems at the intersection of business and IT [66]. Design science research was initially developed for large multi-paper research [36]. For this research, an adapted version was used for smaller research.

The approach used in this paper is based on the standard design science methodology as described in [36], [66] and [95]. As no consensus has yet been reached on an exact methodology, phases were used as a guideline and adapted where it seemed fit. The phases that were followed are:

1. *Problem identification and motivation.* The problem to be solved is identified and the value to find a solution is justified;
2. *Define solution objectives.* The solution objectives are derived from the problem definition and should be formulated from what is expected to be possible and

feasible;

3. *Design and development*. The proposed solution is designed. The research objectives are captured in this design;
4. *Demonstration*. The created design is demonstrated to show how it solves one or multiple of the identified problems;
5. *Evaluation*. The results produced by the created design are observed and measured. This can both be done individually, or in comparison to other designs. Knowledge of relevant metrics and analysis techniques is required in this phase;
6. *Communication*. The created design and the problem it solves are communicated. The novelty, utility and effectiveness of the design are also described.

4.2. Dataset Selection

In order to design a benchmark representative of real-world scenarios and strain, a rich and fitting dataset should be used. In this section, the requirements for selecting a dataset are shown. A selection of datasets were evaluated to these requirements and the dataset deemed most fit was selected. An elaboration on the chosen dataset is provided. Additional details on the dataset can be found in Section 5.2 and Appendix B.

As the goal of this research is to design a benchmark for probabilistic databases, a suitable dataset should be found. For this research, a dataset with the following characteristics is required:

1. The dataset is a good representation of the real world, both in the type of data and in size.
2. The dataset contains enough uncertainty to be suitable for data integration purposes.
3. The dataset should be freely available.
4. The dataset should be versioned. Experiments conducted on the dataset should be reproducible.
5. The dataset is suitable to be inserted in a relational database management system.

To verify which datasets are commonly used for entity resolution, several researches have been analysed. The datasets used in these researches can broadly be categorised in two types: self-collected datasets, as used in [1, 2, 27, 52], and existing datasets, as used in [57, 72, 96, 100].

Regarding the self-collected datasets, it was found that they are all on the small side. Although all freely available, they only contain product data from up to two different websites. Therefore, these datasets are not suitable for this research as they do not meet requirement 1. It also shows that creating a dataset for the purpose of this research is not feasible.

From the existing datasets, the Web Data Commons dataset was most frequently used [72, 96, 100]. Other datasets found in the selection of researches include the Yahoo's

Gemini Product Ads dataset [72], UCI datasets [57], and the LEAPME-dataset [10].

When evaluating these datasets according to the listed requirements, the Yahoo dataset and the LEAPME-dataset are not fit for this research. The Yahoo dataset does not meet requirement 4, as the data can only be retrieved from the API, which always retrieves the most current data. The LEAPME-dataset contains data on four different product categories, which are cameras, headphones, phones and televisions. As all products lie in either of these four product categories, this dataset is not very strong for both requirements 1 and 2.

Both the Web Data Commons (WDC) dataset and several UCI datasets meet all requirements. As the UCI datasets are focused on machine learning and the suitable data sets are smaller than the WDC dataset, the WDC dataset is deemed superior to the other datasets for this research. Datasets from Kaggle [45] were also evaluated, but none were deemed more fit than the WDC dataset.

As the Web Data Commons dataset is provided with a clustering of the products, it is interesting to see whether the probabilistic approach presented in this research will show a clustering similar to that present in the dataset. The WDC Gold Standard also aids this goal. For this standard, a set of 2200 pairs of offers were manually verified whether they belonged to the same product or not [67]. This part of the dataset aided as the ground truth for the machine learning steps taken next.

As it is not the goal of this research to provide a full overview of datasets suitable for entity resolution research, and as no survey paper was found on this topic, we are content with the use of the WDC dataset as it fully meets all requirements. Please note that any other dataset that meets the requirements could have also been used for this research.

4.3. Product Matching

As the area of product matching has been widely studied, this research made use of techniques and approaches that have been described in these earlier researches. The approach relevant to this research is described in this section.

In general, the process of product matching follows the following steps [58, 63, 72]:

1. *Data Preparation.* The data is standardised and cleaned. A uniform data structure is applied.
2. *Search Space Reduction.* Since the time needed for evaluating all possible combinations grows exponentially with the dataset size, the search space for possible matches needs to be reduced to allow for efficient matching.
3. *Attribute Value Matching.* The similarity of the remaining data tuples is determined using a syntactic and semantic means, which produces a comparison vector per data attribute.
4. *Classification.* A decision model then determines the similarity score of a data tuple. This score is compared to the set thresholds to determine whether it is a matching tuple, possibly matching tuple, or non-matching tuple.

5. *Verification*. The performance of the applied product matching algorithm is verified using standard performance metrics.

To prepare the dataset that will be used for the benchmark, these steps were followed. These steps are implemented in QuestionMark: The Dataset Generator [98] and can be referenced for further details. The next part explains the product matching steps that are implemented in this program in more detail.

Data Preparation. A standardised and cleaned version of the chosen dataset is available for download. The quality of this dataset is deemed largely sufficient and no attempt will be made to improve on this or replicate this. As the dataset comes with a clustering, the cluster attribute was removed from each product record to obtain a non-clustered list of product offers.

To allow for variation in the dataset size, a dataset resize function was implemented. This function receives a percentage as an input and pseudo-randomly chooses offers to include in the smaller dataset. For a specified percentage, the same dataset gets returned every time to ensure reproducibility of the results in this research.

Search Space Reduction. To obtain a time-efficient product matching, the search space for matching pairs should be reduced. Disregarding this step results in quadratic time complexity during the product matching phase [52, 64]. Having 16 million products in the dataset, this step is thus essential for a time-efficient product matching. For this step, *filtering* or *blocking* can be used. With blocking, all possible matches are included in the same block and only products contained in the same block are compared [49, 52]. With filtering, a list of possible matches is determined based on a simple similarity measure and a corresponding threshold [64]. A combination of both approaches could also be used.

The surveys of [52] and [64] give an overview of various Filtering techniques, Rule-Based Blocking techniques and Machine Learning Based Blocking techniques. The decision for the space reduction technique used in this research is based on these surveys. Due to time restrictions, a selection of two Rule-Based Blocking techniques was implemented on the dataset to verify which algorithm performed best. These are Incrementally-Adaptive Sorted Neighborhood [97] and Improved Suffix Array Blocking [25]. The literature study showed that these two techniques provide the best performance while still being simple. An implementation of the two blocking algorithms and a performance evaluator was implemented to see which blocking algorithm would be used for this research.

Tests executed using the implemented performance evaluator indicate that Adaptive Sorted Neighborhood is the best blocking algorithm for the dataset used in this research. From the implementation, it is presumed that the Improved Suffix Array blocking technique would perform better when processing complete data consisting of small strings.

Following the method presented by Yan *et al.* [97], the blocking algoritm uses a sliding window to roughly determine what offers are possible matches. For this, a sorted dataset is required. During each iteration of the algoritm, a block is created. The sliding window is placed at the first offer from the sorted list that is not yet in a block.

When the start of the window is set, the enlargement phase is entered. During this phase, the window will iteratively increase in size. This is a fixed size. After each iteration, the blocking algorithm determines the similarity score of the first and last offer in the window. If the distance between the two offers is smaller than the set threshold, the window is enlarged and a new similarity score is determined. If the distance is higher, the retrenchment phase is entered. During the retrenchment phase, the sliding window will decrease one offer in size and calculate the similarity score between the first offer in the window and the new last offer. Once the similarity score rises above the threshold, the block is created.

Attribute Value Matching. As with the search space reduction phase, this phase has a range of implementation options. During this phase, either an algorithm based approach or a machine learning based approach could be used [14, 17]. When product matching with either approach, the matching can be performed only on the product title or on all available information, i.e. including the product attributes [1]. Only using the product title provides simplicity and speed, but at the cost of a lower precision.

As no survey papers were found that discuss this subject specifically, the paper of Bhattacharya and Getoor [14] will be used as an implementation guide. From this paper, the Attribute-Based Entity Resolution approach is used as the foundation of the implementation. For this research, a comparison vector is generated from all attributes of an offer. Within each block, all possible offer combinations are generated and the distance between these offers is then provided by the vector. For simplicity, this vector is combined to a single distance score. The weight of each attribute for this final score is adaptable.

Classification. As the benchmark designed in this research is based on probabilistic data, an additional layer needs to be build on top of the basic algorithm to include a probabilistic model in the final clustering. This implementation will be based on the papers of Panse *et al.* [62] and Elmagrmid *et al.* [28].

The creation of the various probabilistic clusters is based on the possible worlds model as presented in [62]. For each block, a matching graph is created and the matching score of each edge is evaluated. Blocks containing only a single offer are always true, thus they are submitted to a cluster directly. When a block contains multiple offers, their matching score is evaluated. Here exists three possibilities:

1. the matching score of their edges all lie above the upper threshold;
2. the matching score of their edges all lie below the lower threshold;
3. there are one or multiple edges between the two thresholds.

In the first case, the cluster is certain; there is only one possible world. In this case the full block becomes a new cluster. There does exist uncertainty between the correct value of the attributes, as these are likely different. In the second case, the cluster is also certain, as all offers are certainly different. In this case, each offer is put in a separate cluster. In the final case, world graphs should be constructed of the possible worlds. The amount of possible worlds created equals 2^n where n equals the amount of offers connected by an uncertain edge. When these world graphs are created, the inconsistent worlds are removed and the remaining worlds are included as different options for the same cluster. If an offer is certainly present in all worlds, this offer is added later to all generated world graphs. If an offer is certainly not present in all worlds, a separate cluster is created. In these clusters, there exists two kinds of uncertainty. The first kind is the uncertain matching of product offers, which is represented as the probability of the possible world. The second kind is the uncertainty regarding the attribute values of the offers. Both are of importance when working with the probabilistic database.

Uncertainties within a possible world are expressed using probabilities. These probabilities are expressed as values on the unit interval. The probability of correctness of the attribute values of a specific offer within a specific possible world is calculated using the distance measure. For possible worlds with three or more offers, the distance score between the offers is used as a likelihood that the offer has the correct attribute values. When one offer lies closer to all other offers in the cluster, it is likely that it carries the correct information. An offer that lies far from all others likely carries incorrect or incomplete information. For possible worlds containing only two offers, each offer is given the attribute probability of 50%. Regarding the probability of the possible world itself, also the method described by [62] is followed. The probability of a possible world is obtained by the weight of the edges and absence of edges in a possible world. These weights are again determined by the offer distance. Inconsistent worlds are then removed, and the probability of the remaining worlds are normalized.

For the creation of the possible worlds, a naive implementation is used based on the theory presented in this paper. Because of that, the space complexity for the creation of the possible worlds is factorial. This imposes a limit on the block size that can be digested by this algorithm. For this research, this limit is set on a maximum of six offers per block.

Verification. To verify the performance of the blocking and matching algorithms, performance evaluators were implemented. To provide information on the performance, the Gold Standard of the WDC dataset is used. As the structure of this dataset deviates from that of the Offer Corpus dataset, first a transformation on the structure is performed. Additionally, the labels present in the dataset were extracted to test the performance with.

For measuring the performance of the blocking algorithm, the standard performance metrics recall, precision and runtime were used. For the matching algorithm, probabilistic performance measures were used. As the data produced by the matching algorithm is uncertain, standard performance measures will fail to provide a correct representation of the performance. Hence, a selection of the performance measures from [87] was selected. From this paper, the expected precision and expected recall is used, as well as the runtime. In this context, expected recall is defined as the probability mass of the correct answers with respect to the maximum number of correct answers possible. Expected precision is defined as the ratio of probability mass of the correct answers with respect to the probability mass of the complete result set.

For both the blocking and matching algorithm, the results of a selection of tests can be found at Appendix C. These results can act as a guide when selecting the parameters and algorithms during benchmarking, to obtain a dataset with the desired uncertainty and error-rate.

4.4. Query Selection

To obtain a complete and fair set of queries a query selection process was applied. To ensure that the final set of queries does not favour one probabilistic DBMS over the other, queries were designed from the possibilities of the dataset and from the range of operations supported by PostgreSQL.

To obtain a representative set of queries, first the dataset was analysed to verify what useful information could be extracted from the data. The dataset was approached from different business processes for which the dataset could be used. From these business cases, query types were defined. Next, the formulated set of queries was related to the set of operations supported by MayBMS and DuBio. Earlier research on database benchmarking and probabilistic databases was also consulted to obtain useful query types. From the defined query types, the literature study and the supported functionality analysis, a comprehensive set of queries was defined. Finally, this set was reduced to obtain a well balanced set of queries where each functionality is represented equally well.

During benchmark testing, it was discovered that some queries did not provide the insights that were expected of them. This feedback was also used to finalize the set of queries as presented in this paper.

4.5. Metric Selection

The metrics that are included in the benchmark are obtained by a combination of literature research and dataset analysis. Information from Section 3.2 was used as a guide for the collection of metrics. The possibilities of the dataset were evaluated afterwards to obtain an even more complete set of metrics. Only metrics were included that were deemed useful for the evaluation of a DBMS. Therefore, metrics as power consumption are left out. For the metric of user friendliness, additional literature was consulted. Inspiration for statements was drawn from [38] and [39]. The scoring system is based on the Likert scale [53]. A five-point Likert scale is used.

4.6. Executing the Benchmark

To verify the validity of the designed benchmark, it was put to the test with the probabilistic database management systems DuBio and MayBMS. Running the benchmark test on both these systems also provided information on their performance. This contributed to the second goal of this research; providing information on the performance of the tested systems in the context of product matching.

To properly benchmark both systems, a case study was designed. An imaginary business was set up and their requirements and wishes for a probabilistic DBMS were determined. During the benchmarking process, the steps in the manual were directly followed. The results of the benchmarking process are also reflecting the requirements of that business.

For each technology, the same benchmarking process was applied. The designed benchmark was used to benchmark each technology separately. The SQL pseudocode was translated to the corresponding dialect and the parameters were adapted to best fit the system being tested. Each system was deployed in a separate Docker container, all run on the same hardware. The hardware is a single machine with AMD Ryzen 5 PRO 6650U CPU @ 2.90 GHz, quad core, 16 GB RAM, and 512 GB SSD PCIe Gen4. Best efforts were made to make the benchmarking process as fair as possible.

The data collection in this second part of the research is of a quantitative nature. The results are obtained by running the benchmark designed in this research on each of the technologies.

The produced results have been analysed following the method described in the user manual of the designed benchmark, of Appendix A. The parameters used for testing and the results produced are presented in Section 6.1.

4.7. User Testing

To verify the usability of the designed benchmark, user testing was applied. The user study was performed individually and consisted of a formal experiment and an interview. The design of this user study has been approved by the Computer & Information Sciences Ethics Committee of the University of Twente [84].

For this study, research participants were selected who did not have a thorough understanding of probabilistic database technology and benchmarking. According to [12] and [32], more problems get identified by users who have no expert understanding. The participants of this research had to have a good understanding of English and had be proficient with Python. Fitting subjects were asked to participate, so no random selection was used. A total of six subjects participated in this research. From these, five were male and one was female. The participants were between 19 and 26 years old. Participants are following or have completed a University Bachelors or Masters degree in the areas of Computer Science, Electrical Engineering, Applied Mathematics or Interaction Technology.

Before participation to the research, all participants were informed on the purpose of this research and of the possible risks, both written and verbally. The form provided to the participants can be found in Appendix D.1. The users were asked for their consent also both written and verbally.

The first part of the user study consisted of a formal experiment. After the consent was collected, participants were informed on the setup of this experiment. It was explained to the participants that they should imagine that they work in a company as a database administrator and that their manager got instructed to explore the possibilities of implementing probabilistic database technology in the company. Therefore, the manager instructed the team to run a benchmark test on the selected probabilistic database management system using QuestionMark. Their manager left them with the webpage of the benchmark and informed them that the system administrators already set up a working connection to the probabilistic DBMS.

Starting the experiment, the participants were asked to download the benchmark from

GitLab and fully run it. The exact instructions provided to them can be found in Appendix D.2. During this experiment, the researcher remained in the same room as the participants. Participants were allowed to ask questions and were encouraged to share their thoughts.

During the formal experiment, most attention was paid to the rule-based level of performance of the participants. With the rule-based level of performance, errors often correspond to omitting steps in the procedure, either due to participants missing verbal or non-verbal cues about critical steps in this procedure. Problems on this level are caused by issues in cuing, or consistency of the software [32]. On this level, the ergonomic quality of the software could best be tested. The ergonomic quality is related to the usability of the software in terms of its simplicity, controllability and predictability [39]. Finally, most usability problems are discovered when combining both usability testing and heuristic evaluation [32]. Therefore, during user testing, special attention was paid to the ten heuristics of Nielson-Molich [60]:

- Keep users informed about its status.
- Show information in ways users understand.
- Offer users control.
- Offer consistence in interface.
- Prevent errors.
- Have visible information and instructions to let users recognize options and actions.
- Be flexible so experienced users can reach the goal faster.
- Have no clutter.
- Provide plain-language help.
- List concise steps in clear documentation.

After the participant has fully completed the benchmark process, an interview was conducted. The following questions were asked to the participants:

1. What do you think?
2. How would you rate the user manual provided with the benchmark? Did you have trouble understanding specific steps? Were steps missing?
3. How do you rate the user-friendliness? Was it easy or difficult to use?
4. How would you rate your own Python level? Was that sufficient to run the benchmark?
5. How much previous knowledge on benchmarking and (probabilistic) database technology do you have?
6. Can you understand the provided results?
7. Do you have any further suggestions for improvement?
8. Anything else you would like to add?

These questions were collected by using insights from [75], [79] and [81]. The interview questions were mostly open-ended to allow for follow-up questions and encourage additional explanation by the participant.

The results of this study were used to iterate on the design of the benchmark. These results can be found in Section 6.2.

5

Benchmark

In this chapter, the probabilistic benchmark design will be presented. This section touches upon the final design decisions, provides the queries that are included in the benchmark and explains how the benchmark can be adapted to approach the load of the real-world applications. This section provides the required knowledge about the benchmark and provides a guide how the benchmark should be used. The benchmark manual can be found at Appendix A.

5.1. The QuestionMark Benchmark

The benchmark presented in this thesis is called QuestionMark. The QuestionMark benchmark for probabilistic databases is a benchmark system is composed of two Python programs that both need to be executed for using the benchmark. In short, the two programs are:

- *QuestionMark: The Dataset Generator*. This program generates the dataset required for running the actual benchmark. By tweaking the parameters in this program, a dataset can be generated that approaches any real-world scenario as closely as possible. This Python program can be downloaded from GitLab [98].
- *QuestionMark: The Probabilistic Benchmark*. This program can run the probabilistic benchmark presented in this research. The program can be adapted to benchmark any probabilistic DBMS. This Python program can also be downloaded from GitLab [99].

As there is no standard available for benchmarking probabilistic databases, this research aimed to deliver a benchmark that covers a wide range of aspects of the tested probabilistic DBMS. The design of this benchmark is based on the foundations presented in Section 3 and designed following the principles of design science. This benchmark provides a convenient way to test various probabilistic database management systems and get insights on their performance. Since the queries provided in this benchmark are written in a pseudocode like language, queries can easily be translated to any probabilistic dialect. Additionally, it provides clear guidance on how the parameters can be adapted to approach any real-world application as close as possible.

The processes performed by both programs are visually displayed in figures 5.1 and 5.2. These figures are based on the UML Activity Diagram. In these figures, an orange coloured box indicates that the process requires human action. All white boxes are thus performed automatically.

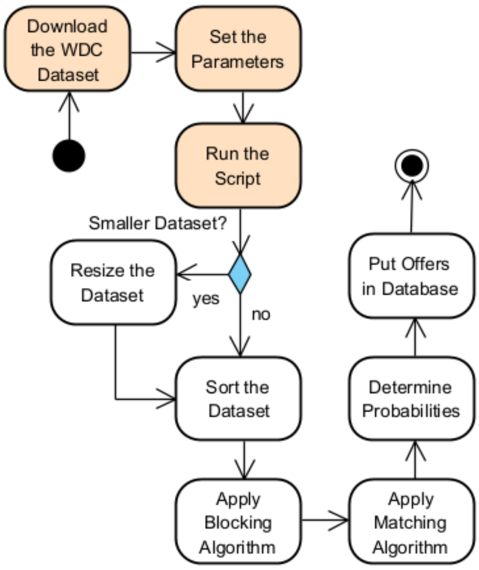


Figure 5.1: The processes performed by QuestionMark: The Dataset Generator

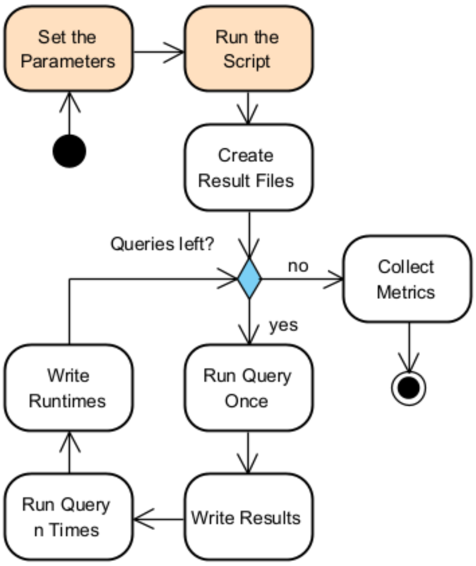


Figure 5.2: The processes performed by QuestionMark: The Probabilistic Benchmark

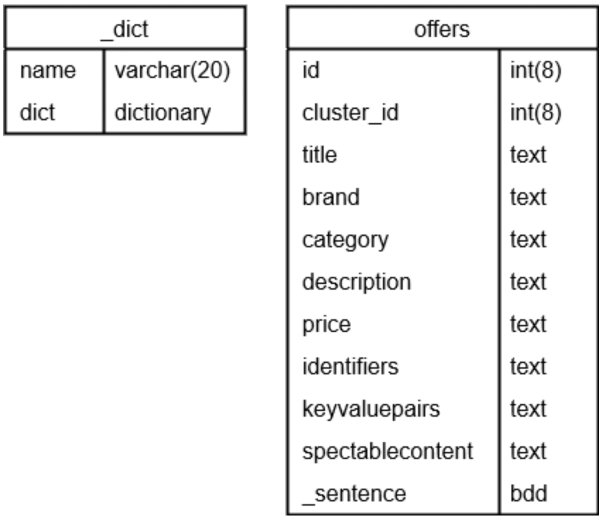
5.2. Dataset

This benchmark uses an adaptation of the WDC Product Data Corpus and Gold Standard for Large-Scale Product Matching, Version 2.0 [93] as the dataset. From this data corpus, the normalised English offers dataset is adapted and used for this benchmark.

The WDC dataset is a large public training dataset for product matching. It is produced by extracting schema.org product descriptions from 79 thousand websites, which provides 26 million product offers. The English offers subset consists of 16 million product offers [67]. The dataset is provided with a clustering. The 16 million product offers in the English subset are categorized in 10 million clusters. Each cluster contains offers of the same product found on different websites. There are roughly 8.5 million clusters with size 1, one million clusters with size 2, and 400.000 clusters with size 3 or 4. Clusters of a size greater than 80 are filtered out of the dataset, as these are likely noise [67]. Within the English offers dataset, each offer is represented as a JSON object. An example offer can be found in Appendix B.2.

To obtain a dataset suited for use in this benchmark, the English offers subset from the WDC data corpus is adapted. The dataset was adapted to include a probabilistic clustering. The methodology used for this can be found in sections 4.2 and 4.3.

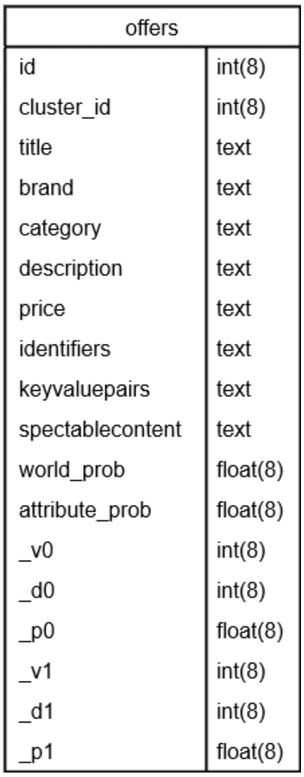
The final dataset structure obtained is shown in Figure 5.3 for DuBio and in Figure 5.4 for MayBMS.



_dict	
name	varchar(20)
dict	dictionary

offers	
id	int(8)
cluster_id	int(8)
title	text
brand	text
category	text
description	text
price	text
identifiers	text
keyvaluepairs	text
spectablecontent	text
_sentence	bdd

Figure 5.3: The dataset structure in DuBio



offers	
id	int(8)
cluster_id	int(8)
title	text
brand	text
category	text
description	text
price	text
identifiers	text
keyvaluepairs	text
spectablecontent	text
world_prob	float(8)
attribute_prob	float(8)
_v0	int(8)
_d0	int(8)
_p0	float(8)
_v1	int(8)
_d1	int(8)
_p1	float(8)

Figure 5.4: The dataset structure in MayBMS

5.3. Parameter Tuning

The benchmark is developed to support a wide range of strain levels. Both Python programs contain parameters that should be tweaked before running. During the dataset generation phase, there are multiple parameters that can be tweaked to change the size and uncertainty of the dataset. These parameters should be tweaked to represent the real-world data as closely as possible. During the benchmarking, parameters can be tweaked to alter the coverage of queries and change the behaviour of the benchmarking. The following parameters are present in QuestionMark: The Dataset Generator:

- *DBMS*. Determines into which database management system the generated dataset should be loaded and what preparatory queries need to be run.
- *Dataset size*. Determines the amount of offers included in the dataset. A percentage of the dataset can be determined with up to two decimal places. The offers for the new smaller dataset are pseudo-randomly chosen, so that the same dataset is returned for multiple runs. This ensures reproducibility of the results. The full dataset contains 16 451 499 offers. The smallest dataset that can be generated is 0.01% of the full dataset, which produces an initial dataset of 1653 offers. The final probabilistic dataset that is generated from these offers contains 11 807 records. This value should be carefully chosen, as this influences to what extent the produced dataset imitates the data being digested by the real-world application.
- *Whole clusters*. Determines whether the offers chosen from the larger dataset

to include in the new smaller dataset are pulled from entire clusters or not. Including entire clusters increases the uncertainty of the data.

- *Word distance measure.* Determines the way the distance between two words or sentences is calculated. This measure is used during the blocking phase on the attributes determined as Blocking Key Values and on all suitable attributes during the matching phase. The implemented distance measures are Levenshtein, Jaro, Jaro-Winkler, Hamming and Jaccard.
- *Blocking key values.* Determines the attributes that are included to determine the similarity of two offers during the blocking phase. Including more attributes provides a better blocking performance, but at the cost of a higher run time.
- *Blocking similarity threshold.* Value between 0 and 1 that represents the distance between two offers. Evaluated offers with a distance lower than the threshold are included in the same block.
- *Blocking window size.* Determines the size of the sliding window. Within a window, the distance between the first and last offer is determined. This value influences the run time.
- *Maximum block size.* Poses a restriction on the block size. Increasing this value improves the performance. As the matching phase includes a calculation with factorial time complexity, this size should not exceed seven. Six is advised.
- *Matching attributes.* Determines the attributes that are used to determine the distance between two offers during the matching phase. Including more attributes improves the performance, but increases the run time.
- *Matching attribute weights.* Determines the weight of each attribute to calculate the final distance score. This can be tweaked to improve the performance. It has no effect on the run time.
- *Upper phi and Lower phi.* Determines the upper and lower threshold of the distance measure. If the distance between two offers is greater than the upper phi, the two offers are certainly not the same product. If the distance is smaller than the lower phi, the two offers are certainly the same. Increasing the gap between the values ensures less false matches or non-matches, but increases the computational complexity in later phases and during querying. A smaller gap can be used to artificially reduce the uncertainty in the dataset. This value should be carefully chosen, as this influences to what extent the produced dataset imitates the data being digested by the real-world application.

The following parameters are present in QuestionMark: The Probabilistic Benchmark:

- *DBMS.* Determines the Database Management System that will be used for the execution of the benchmark. Additional systems can be added when support for them is also added to the benchmark program.
- *Iterations.* Denotes the amount of times a query is run to obtain a runtime average from the queries. This is a global variable that is used for all queries. Increasing this number will provide a more precise outcome of the average run

time, but at the cost of a longer benchmark execution time. The total amount of iterations is always +1 to create a warm start.

- *Show Query Plan*. Boolean value. If true, the query plan for each query is also provided with the benchmark result. Enabling this variable does not influence the execution time of the queries.
- *Timeout*. Ensures that queries that take too long to return an answer will be aborted. Once a query times out, the benchmark run will be quit and an error message will be displayed.
- *Queries*. A list that contains all queries from the benchmark. Depending on the goal with which the benchmark is run, queries that are not relevant can be removed from the benchmark run. Removing queries lowers the total time required to run the benchmark.

5.4. Queries

This benchmark offers a total of eighteen queries. These are a range of queries that can be used to test various types of systems. These queries cover a wide range of functionalities. Queries for the benchmark are divided into three categories:

- Queries providing insights into the dataset.
- Queries that perform probabilistic operations on the dataset.
- Queries that insert, update or delete records or uncertainty.

In the benchmark, a distinction is made between native support of a function or the possibility to include the function, for example by using a workaround method. Table 5.1 lists the functionalities covered by the benchmark test and the queries that test for that support. The twelfth functionality regarding the anomalies is included as an addendum for functionalities that are not listed, but turn out to either be lacking from the tested system or be included.

#	**Functionality**	**Queries**
1	Support of most recent deterministic DBMS queries	Any
2	Offering a compact representation of the present uncertainty	Insight 2
3	Get the probability of an offer	Probabilistic 1
4	Get the probability of a composed result	Insight 5; Insight 6; Probabilistic 4
5	Apply aggregate functions on probabilities	Insight 4; Probabilistic 4
6	Filtering on probability	Probabilistic 6
7	Get the expected count	Probabilistic 2
8	Get the expected sum	Probabilistic 3
9	Get the most probable answer	Probabilistic 5
10	Verify if a specific possible world exists	Insight 5
11	Verify if a record is certain	Insight 4

12	Updating the uncertainty of an offer	Insert, Update, Delete 3
13	Repair the probability space after addition, update or deletion of offers	Insert, Update, Delete 1; Insert, Update, Delete 4
14	Any anomalies discovered during benchmarking	Any

Table 5.1: Functionalities and Query Support.

For each functionality, a selection of queries is offered that can be selected and run separately. This provides two benefits. First, the wide range of queries ensures that there are always some queries that represent the load of the real-world application for which the benchmark is run. Second, this ensures that only the queries that represent the load of the real-world application need to be run, thus saving time.

The queries for this benchmark are provided in a pseudocode like SQL-dialect. Since the different probabilistic databases have distinctive specialised dialects, a generic language provides convenience to translate to the desired dialect. To serve research purposes, the pseudocode queries have been translated to the dialects of MayBMS and DuBio. The queries included in the benchmark are listed below. The translation of these queries in both dialects can be found in Appendix A.7.

The queries below are included in the benchmark. For each query, additional information is provided on its functioning and why it is included in the benchmark.

Test 1: Testing the connection. The first query is mainly included to have a low strain query that can be used to test the connection. This query consists of basic SQL-functionalities and all systems should be able to run this.

```
select attribute 'id'
from entity 'offers'
return the first 10 records;
```

Insight 1: Retrieve the full dataset, gain insight in data structure. This query is a real strain tester of the system. The query itself is simple, but it requires the DBMS to return all its data.

```
select all attributes
from 'offers';

if present select all remaining data;
```

Insight 2: Provide insight into the concentration of offers. This query can be used to verify to what extent the DBMS can concentrate the uncertainty of an offer. It also provides insights into the number of clusters that have been formed.

```
select the count of all attributes alias 'records',
   ↪ the count of all distinct values of attribute 'id' alias 'offers',
   ↪ the count of all distinct values of attribute 'cluster_id' alias '
   ↪ clusters'
from entity 'offers';
```

Insight 3: Provide insight into the distribution of cluster volumes. This query is included as lower strain deterministic query and also includes useful insight into the dataset. As larger clusters put more strain on probability calculations, it is useful to gain insight into the distribution of cluster volumes.

```
select attribute 'cluster_size',
   ↪ the count of all values of attribute 'cluster_size' alias 'amount'
from subquery (
    select the count of all distinct values of attribute 'id' alias '
       ↪ cluster_size'
    from entity 'offers'
    grouped by attribute 'cluster_id'
) alias 'cluster_sizes'
grouped by attribute 'cluster_size'
ascendingly ordered by 'cluster_size';
```

Insight 4: Gets the percentage of certain clusters. This query provides insight into the uncertainty of the generated dataset. A new dataset can be generated when the result of this query does not match the uncertainty of the real-world dataset. It also verifies if probability calculations can be done on aggregated data.

```
select the count of all certain records divided by the count of all
   ↪ attributes times 100 rounded to four decimal places alias 'certain
   ↪ percentage'
from entity offers;
```

Insight 5: Get the id and probability of the offers from a specific possible world. In some situations it might turn out useful to make a selection based on the probability space of a record. This query returns any record satisfying a specific sentence or probability space declaration.

```
select attribute 'id',
   ↪ the probability attribute,
   ↪ the variable or sentence attribute
from entity 'offers'
satisfying a specific variable or sentence statement;
```

Insight 6: Get the average probability of the dataset. This is another query that tests the strain of the system. It performs a probability calculation over the entire dataset. It also provides insights into the uncertainty of the dataset.

```
select the average of the probability attribute rounded to four decimal
   ↪ places alias 'certainty_of_the_dataset'
from entity 'offers';
```

Probabilistic 1: Get offers with the probability of their occurrence. This query contains the most basic added functionality of any probabilistic DBMS, which is the presentation of the probability. It also evaluates the speed of ordering based on the probability attribute.

```
select the probability attribute rounded to four decimal places alias '
   ↪ probability',
   ↪ all attributes
from entity 'offers'
descendingly ordered by 'probability';
```

Probabilistic 2: Gets the expected count of the categories. One more advanced operation on probabilistic data is to obtain the expected count of an attribute. This query evaluates if that operation is supported.

```
select the attribute 'category',
   ↪ the expected count per attribute 'category' alias 'expected_count'
from entity 'offers'
grouped by attribute 'category'
descendingly ordered by 'expected_count';
```

Probabilistic 3: Gets the expected sum of the product ids per cluster. Another closely related operation is the expected sum. This query evaluates if that operation is supported.

```
select attribute 'cluster_id',
   ↪ the expected sum per attribute 'id' alias 'number_of_offers'
from entity 'offers'
grouped by attribute 'cluster_id'
descendingly ordered by 'number_of_offers';
```

Probablistic 4: Gets the variables/sentence and probability for the categories. This query is again focused on strain testing. This query produces large aggregations of probabilities, which need to be evaluated to return the query result. This query tests if the DBMS can digest these large aggregations.

```
select attribute 'category',
   ↪ the compound variable/sentence attribute,
   ↪ the compound probability attribute rounded to four demical places
   ↪ alias 'probability'
from entity 'offers'
grouped by attribute 'category'
descendingly ordered by 'probability';
```

Probabilistic 5: Returns the most probable offer that is related to a specified string. This query represents the behaviour of a search engine, where the most probable offer satisfying a search condition should be returned. An example string is 'card'. The pseudocode provided contains a workaround method to obtain the most probable answer. It can be shortened to represent native support of this functionality. This query contains hard-coded information in the translation and may require an adaptation when having generated a fitting dataset. See Section 5.4.1 for more information.

```
select all attributes,
   ↪ the probability attribute rounded to four decimal places alias '
   ↪ probability'
from entity 'offers'
satisfying that the attribute value 'cluster_id' exists in subquery (
        select attribute 'cluster_id'
        from entity 'offers'
        satisfying that attribute 'title' contains a specified string
                or that attribute 'description' contains a specified
                   ↪ string
)
descendingly ordered by 'probability'
return the first 1 records;
```

Probabilistic 6: Returns all offers containing a specified string with a high uncertainty. When a dataset contains large volumes of highly uncertain data, it can be useful to let a selection of data pass human inspection. This query returns the most uncertain offers so these can be manually classified. This query contains hard-coded information in the translation and may require an adaptation when having generated a fitting dataset. See Section 5.4.1 for more information.

```
select attribute 'id',
   ↪ attribute 'cluster_id',
   ↪ attribute 'brand',
   ↪ attribute 'category',
   ↪ attribute 'identifiers'
from entity 'offers'
satisfying that attribute 'title' contains a specified string
        or that attribute 'description' contains a specified string
        and the value of the probability attribute is higher than 0.45
        and the value of the probability attribute is lower than 0.55;
```

Insert, Update, Delete 1: Inserting a new probabilistic cluster. When dealing with probabilistic databases, new data can be added regularly. This query verifies the speed at which new clusters can be added to the database.

```
insert into entity 'offers'
the values (a copy of a cluster with size 5, with negative id values.);

if required add the new probabilities to the corresponding entity;
if required manually repair the probability space;
```

Insert, Update, Delete 2: Inserting bulk. When large volumes of data are constantly added to the database, they are likely added in bulk. This query strain tests the DBMS on large additions of probabilistic data. The current table 'bulk insert' contains 1000 offers and their corresponding probabilities.

```
insert into entity 'offers'
the results of subquery (
        select all attributes
        from entity 'bulk_insert'
);

if required add the new probabilities to the corresponding entity;
if required manually repair the probability space;
```

Insert, Update, Delete 3: Update uncertainty. This query updates the uncertainty of a specific cluster. As the location of the probability greatly determines the form of this query, its pseudocode is more abstract. This query contains hard-coded information and may require an adaptation when having generated a fitting dataset. See Section 5.4.1 for more information.

```
update the entity containing the probabilities.
alter half of the probabilities of a cluster with four offers;

if required manually repair the probability space;
```

Insert, Update, Delete 4: Remove uncertainty. When working with probabilistic data, chances are that new evidence will be found and the database should be updated accordingly. In this query, a cluster of size four will be split into three clusters. The translation of this query contains hard-coded information and may require an adaptation when having generated a fitting dataset. See Section 5.4.1 for more information.

```
update entity 'offers'
set attribute 'cluster_id' with the maximum value of attribute 'cluster_id
    ↪ ' + 1,
    the variable/sentence/probability attribute to certain
satisfying that attribute 'id' has the value of the first offer in the
    ↪ cluster;

update entity 'offers'
set attribute 'cluster_id' with the maximum value of attribute 'cluster_id
    ↪ ' + 1,
    the variable/sentence/probability attribute to certain
satisfying that attribute 'id' has the value of the third offer in the
    ↪ cluster;

update entity 'offers'
set the variable/sentence/probability attribute to a new normalized value
satisfying that attribute 'id' has the value of the second offer in the
    ↪ cluster;

update entity 'offers'
set the variable/sentence/probability attribute to a new normalized value
satisfying that attribute 'id' has the value of the fourth offer in the
    ↪ cluster;

if required update the probabilities in the corresponding entity
if required update the probability space;
```

Insert, Update, Delete 5: Delete a full cluster. Any probabilistic data should also not slow down the deletion of data significantly. This query tests the speed of the DBMS when deleting probabilistic data. This query should be run on a cluster with size four. The implementation of this query contains hard-coded information and may require an adaptation when having generated a fitting dataset. See Section 5.4.1 for more information.

```
delete all records from entity 'offers'
satisfying that attribute 'cluster_id' has the value of the specified
    ↪ cluster;

if required delete the probabilities in the corresponding entity;
if required manually repair the probability space;
```

5.4.1. Altering Queries

Some of the queries presented above contain variable values. As the offer included in the dataset varies when a dataset is created with a different size, the values included in the dialect translations can become incorrect. The instructions below explain how each variable query should be adapted for use.

- *Query insight 5.* This query requires a specific variable or sentence to be defined. You could either define one that does not exist in the database, or choose one that does exist.
- *Queries probabilistic 5 and probabilistic 6.* This query uses pattern matching to obtain a selection of offers that satisfy that pattern. It is advised to query for anything that exists in the dataset.
- *Query insert, update, delete 3.* This query requires a specific cluster to be defined. Seek for any cluster of size four. Include its ID in the query and change the probability with variables accordingly.
- *Query insert, update, delete 4.* This query should also be run on any cluster of size four. Include the ID of each offer present in that cluster in one of the four queries. Include the cluster ID in the probability variables.
- *Query insert, update, delete 5.* This query removes a cluster. Search for a cluster with a sufficiently large size and include its ID. With the current limitations, a cluster with the largest size is advised.
- *Queries timing out.* During the benchmarking, it could happen that queries take too long to return an answer. In that case, the query is timed out. To verify whether the functionality of the query is supported, change the query to run on the part table. This part table contains a small portion of the dataset. If the query still times out with this table, it could be worthwhile to decrease the size of this table even further. To do this, go to QuestionMark: The Dataset Generator and open `database_filler_[DBMS].py`. Then reduce the value in `LIMIT FLOOR()` in the first query of `prep_queries`.
- *Queries raising exceptions.* During the benchmarking, it could also happen that queries throw errors. When any query raises the exception `invalid memory alloc request size 1073741824` or `Ran out of memory retrieving query results`, it can also be worthwhile to run the query on the 'part' table. Most likely, reducing the dataset size that the query needs to digest removes this specific error. This verifies whether the functionalities in the query are supported by the system or not.

5.4.2. Query Implementation Decisions

Some of the queries defined in the benchmark were difficult to translate for a specific dialect. During the translation, some decisions were made that deserve some additional explanation. For DuBio, the following queries required a special approach:

- *Insight 6.* DuBio does not have a function to retrieve the average probability over all offer. Therefore, a manual approach was implemented in the query.
- *Probabilistic 2.* DuBio does not have a function to obtain the expected count of a query result. Therefore, a manual approach was implemented.
- *Probabilistic 3.* DuBio does not have a function to obtain the expected sum of a query result. Therefore, a manual approach was implemented.
- *Probabilistic 4.* It was decided to implement a clearer, more wordy implemen-

tation than to make it more compact.

- *Insert Update Delete 1 and 2*. DuBio does not have an easy way to insert all random variables with their corresponding probabilities in the dictionary. This process was manually typed into the query. For 2, the query `SELECT print(dict) FROM bulk_dict WHERE name='mydict';` was run to obtain the values generated by the preparatory queries.

For MayBMS, the following queries required a special approach:

- *Insight 4*. MayBMS does not have a function to retrieve all certain offers. Therefore, a manual approach was implemented in the query.
- *Probabilistic 4*. MayBMS does not support the creation of long composed random variables. This query is practically impossible in MayBMS. Currently, an implementation is chosen without the random variable information. The correct implementation is theoretically possible to obtain. In this scenario, for each category a new table should be created containing only the products from that table. Next, for each created category table, all present offers should be grouped into one offer. As MayBMS does not display the random variable information when a `GROUP BY` is used, the result can be obtained by generating a cross join over all records in the specific category table. The query will thus be: `SELECT category, tconf() FROM category_table c1, category_table c2 ... category_table cn;`, where n is the result of the query `SELECT COUNT(category) FROM offers WHERE category = 'your_category';`. This would create a query that is too large to comfortably work with and is unrealistic to be implemented. It is thus concluded that the functionality is not supported by MayBMS.
- *All 5 Insert Update Delete*. MayBMS does not manually repair the probability space. To ensure a correct representation, the offers table should be build from scratch each time an adaption is made. Therefore, adaptions are made to the offers_setup table, after which the repair keys are created over the world probabilities and attribute probabilities, which are then joined to create a new offers table.
- *Insert Update Delete 3*. As the approach of any dialect can be rather unique, a compact description is provided by the pseudocode. Since MayBMS does not have a compact representation of the offers, multiple records in the database need to be adapted to update the probability space of a single offer.

5.5. Metrics

The QuestionMark benchmark has a selection of metrics that are simple yet powerful. The selected metrics cover the most important aspects of a probabilistic database management system and can provide a good indication of how well the tested system fits the real-world scenario it will be used for. These metrics combined measure the supported functionalities of the system, its speed and the usability. For this benchmark, the following metrics are included.

Query functionality coverage. This metric provides insight into the functionality coverage of the database system and is determined by multiple sub-metrics. When running the queries to obtain their results and runtime, it can happen that a specific functionality is not supported or the database system cannot handle the load required to execute the query. In these cases, the system returns an error. The error raised during execution are stored and printed as the query result. After the benchmark execution has finished, an overview table is created that indicates what queries finished execution and which threw an error. The percentage of successful queries is then also determined. For each query that threw an error, it also indicates what query functionality might be lacking. In each case, a critical look is needed to verify whether the error is thrown due to an actual lack of functionality support or due to another reason, for example a typo. With the gathered knowledge, the functionality coverage table can be manually filled in. In this table, a distinction is made between functionality that is natively supported and functionality that can be implemented with a workaround method

Brevity of the query dialect. This metric is determined by the total amount of characters needed for all queries and gives insights into the succinctness of the query language. A more succinct query dialect often requires less time to write queries with and is often easier to understand. This metric value is obtained by iterating over all queries and adding their character count. Spaces are removed from the calculation. Optionally, characters can be removed from specific queries. For example in query `IUD_1_rollback` offers are added to the database. As the data that represents the offer is not indicative of the complexity of the query language, the amount of characters used for that representation is subtracted from the total character count for that query.

Runtime of queries. This metric provides insight into the speed of query execution. A lower runtime is required to obtain higher query throughput rates and improves the flow of business processes relying on the query results. This metric is also obtained by a combination of sub-metrics. To obtain the runtime of a query, the PostgreSQL `EXPLAIN ANALYSE` statement is used. This statement returns the execution plan of various queries or statements and tracks its runtime. When available, it differentiates between the planning time and execution time of a query. In this distinction is not supported by the DBMS, only a total runtime is returned. For each query, the average runtime over the specified iterations is printed. Each query is run with a warm start. After all benchmark queries have run, a total average planning time and execution time, or total average runtime is calculated. This is the sum of all time averages of all queries. The total time provides a quick idea of the speed of the tested DBMS. For each application scenario, the acceptable runtime of a query differs. It is thus advised to verify the significance of the queries and per query determine the acceptable runtime.

Probabilistic data overhead. This metric represents the additional storage space required to store the probabilistic representation of the data. When processing large volumes of data, needing additional storage space to store the probabilistic representation of the data could get costly. As each probabilistic DBMS stores their probabilistic representation in a unique way, the probabilistic data overload is calculated for each DBMS differently. For both systems, the storage space used is determined by the `pg_size_pretty` statement of PostgreSQL. For DuBio, the overhead percentage is determined using the following calculation:

$$\frac{sentence + dictionary}{offers + dictionary} \times 100$$

Here, *sentence* is the size `_sentence` column in the `offers` table, *dictionary* is the size of the `_dict` table, and *offers* is the size of the `offers` table.

For MayBMS, the following calculation is used to determine the overhead percentage:

$$\frac{setup \times (1 - \frac{distinct\ ids\ count}{ids\ count}) + (offers - setup)}{offers} \times 100$$

Here, *setup* is the size of the `offers_setup` table, *distinct_ids_count* is the count of all distinct values of the `id` column in the `offers` table, *ids_count* is the count of all values of the `id` column in the `offers` table, and *offers* is the size of the `offers` table.

The calculation for MayBMS is a bit more complex, as MayBMS does not create a compact representation of the probability space over a single offer. Because of that, data duplication is created in the offers table. The overhead that this duplication creates is determined by counting the `id` values.

User friendliness. User friendliness is another metric that is composed from several sub-metrics. As user friendliness is something of a more personal taste and cannot be measured from a benchmark run, all sub-metrics are in the form of statements that should be rated on a scale from 1 to 5. On this scale, a 1 means that the user strongly disagrees with the statement and a 5 means that the user strongly agrees. The following aspects should be evaluated to determine a final user friendliness score of the system:

[1, 2, 3, 4, 5]	The software is well documented.
[1, 2, 3, 4, 5]	The software was easy to work with.
[1, 2, 3, 4, 5]	We have sufficient in-house expertise to work well with the software.
[1, 2, 3, 4, 5]	I am satisfied with the monetary expenses that need to be made for running the software.
[1, 2, 3, 4, 5]	The software has a good support service.

5.6. Design Decisions

During the design phase of the benchmark, several measures have been implemented to ensure that this benchmark is well designed and fair. The measures taken are described in this section in alphabetical order.

Benchmark Results Documents. To provide an easy overview of the benchmark results, the produced results are all loaded into two benchmark results document. The first document is focused on providing the results of some generic metrics. The second document provides the outcomes of a selection of metrics per query.

Code Documentation. Code documentation is applied to provide clarity and transparency on the codebase and to guide the user in the use of the benchmark. To ensure the code is easily understandable by any programmer, the code is thoroughly commented. These comments provide a general explanation of the defined functions and provide additional information on more complex code. The code is also accompanied by a compact README and a comprehensive instruction manual.

Dataset Selection. As also explained in Section 4.2, best efforts were made to select a dataset that is a good fit for this benchmark. A set of requirements was defined to obtain a dataset that is true to the real world, can easily be reproduced and is transparent. The chosen dataset meets the set requirements. Nevertheless, the chosen dataset does have some practical limitations. As an existing dataset is chosen, a generation bias towards any technology is not present. Previous knowledge on DuBio was also not used to influence the selection of a dataset.

Dataset Size Generation. The Dataset Generator includes a function that can alter the size of the generated dataset. This size alteration ensures that the load that the benchmark will test approaches the load of the real-world application it has to mimic. As the dataset is originally ordered, including only the first x records is not the best option. To also ensure for reproducibility, a selection of offers or clusters is chosen based on the hash of the id. This provides a pseudo-random selection of offers or clusters. During the dataset resize phase, a choice can already be made between a dataset with higher uncertainty or one with lower uncertainty. Choosing offers based on their ID gives a higher variation in offers, producing more clusters that are smaller. Choosing offers based on their cluster ID selects full clusters of offers, resulting in fewer clusters that are larger. The size of the dataset can be specified as a percentage of up to two decimal places of the full dataset size.

Guided inclusion to support new software. As each probabilistic DBMS has its own unique way of storing and displaying uncertainty, new benchmark code must be added to support this data storing. The benchmark guides in this process by providing step-wise instructions and being well-documented. The benchmark queries should also be translated to the corresponding SQL dialect. Although the code documentation guides the process of adding a new probabilistic DBMS, programming experience is required to do this.

Instruction Manual. With the benchmark, a clear instruction manual is provided. This manual contains information on the functioning of the two programs, on the parameters that can be tuned, and on the queries. It also guides the user in the processing of the benchmark results. The instruction manual explains how the system should be adapted to benchmark new probabilistic database management systems.

Manual Roadmap. To guide the user on how to use the benchmark, each of the two programs is equipped with a comprehensive instruction manual. This instruction manual contains a roadmap on how to use the program out of the box for benchmarking DuBio and/or MayBMS. The manual in natural language is also mirrored in a Python file, where the listed functions can be easily found and run. This ensures that everyone with minor programming experience can work with the benchmark. It also provides instructions on how to adapt the program to include any new probabilistic

DBMS.

Metrics. As this is the first generic benchmark to appear for probabilistic databases, a selection of metrics were chosen that are simple yet powerful to provide a complete picture of the system tested. The metrics included are runtime of queries, operation support, query length, size of the probabilistic representation and usability.

Open Source Codebase. All code from the QuestionMark benchmark is freely available for download at GitLab [98, 99]. This provides transparency in the dataset generation and benchmarking process. Best efforts were made to create a well-documented and well-readable codebase. The QuestionMark benchmark for probabilistic databases is released under the CC Attribution 4.0 International license, also known as CC BY 4.0, for clarity and transparency.

Parameter Tuning. To ensure that the benchmark can be adapted to approach any real-world scenario as closely as possible, various parameters are defined in the benchmark that can be tuned to alter the final result of the dataset and benchmark test. In both benchmark programs, a Python file is included that contains all parameters as global variables. This file also contains an explanation of the present variables to provide guidance on how to tune them. By having all parameters at a central place, they can easily be found and adapted for tuning. Apart from improving the (expected) precision and recall of the dataset, parameters are also present to scale the amount of data included in the dataset and the uncertainty that data has.

Product Matching Algorithms. The selected dataset is clustered with a deterministic approach. To make this a probabilistic dataset, the existing clusters have been removed and a new probabilistic clustering has been generated. For this probabilistic clustering, several algorithms have been used. More information on these algorithms and the selection process can be found in Section 4.3. Introducing a new probabilistic clustering ensures that the dataset approaches any real-world scenario better than by adapting the existing clustering. The existing clustering can be used as a means to determine the new clustering.

Product Matching Performance Evaluators. The dataset produced by QuestionMark: The Dataset Generator can be verified on quality. For this, additional functions have been added to the program that verify the performance in terms of (expected) precision and recall. For this, the WDC Golden Standard dataset is used, which is a small labelled version of the WDC Product Offers dataset. Providing these performance measures provides transparency in the quality of the produced dataset and can be used as an indicator when tuning parameters.

Pseudocode SQL. As this benchmark is designed to cater for multiple systems, a generic manner of displaying the required queries was required. For programming languages, pseudocode can be used to define the steps of an algorithm in natural language. For SQL this standardised method was not yet defined. In this research, a pseudocode like language is defined and used to allow for easy translation of the included queries to any new dialect. This omits the need to specialise in the dialects of MayBMS or DuBio to be able to understand the included queries or add a new dialect.

Real-world Approaching. On the spectrum from synthetic to real world, the QuestionMark benchmark is approaching real world. Earlier research on probabilistic databases tested the developed systems mainly on synthetic data in a synthetic environment. A benchmark that can be tuned to mimic a range of real-world scenarios provides a true image of the performance of any probabilistic DBMS tested.

Two Python Programs. The QuestionMark Benchmark is divided in two Python programs. The decision was made to separate the dataset generation and the execution of the benchmark test. The two functions are sufficiently distinct that the decreased complexity of each program outweighs the benefit of slightly lesser setup time. It provides two simpler instruction manuals and ensures a clear separation of concerns.

Queries. For this benchmark, a range of queries was selected that cover the full functionality desired by any probabilistic DBMS. As the benchmark is designed to test any probabilistic DBMS, queries should be formulated independently from the functionalities of either MayBMS or DuBio. The used methodology can be found in Section 4.4. This methodology results in queries that include the desired functionality of any probabilistic DBMS. This ensures that the queries represent real-world scenarios, have a broad range from targeting specific parts to the full system, and that no existing DBMS is favoured. As the queries are partially based on the known possibilities of the probabilistic database systems MayBMS and DuBio, it could be that the possibilities of another probabilistic DBMS is not fully represented. Users of the benchmark are free to define and add their own queries to the benchmark when functionalities are not yet included.

6 Empirical Evaluation

After the benchmark has been designed, it is time to put it to the test. In this chapter, the benchmark will be run on both MayBMS and DuBio. Simple user tests will also be run with the benchmark. Finally, a conclusion will be drawn on the performance of the presented benchmark.

6.1. Case Study: Benchmarking MayBMS and DuBio

To gain insights into the descriptive capabilities of the benchmark and to put it to the test, both MayBMS and DuBio are benchmarked using the QuestionMark benchmark. The methodology used for this is described in Section 4.6. The performance results of both systems will be presented in this section. The code and queries required for running the benchmark tests on both systems are included in the benchmark codebase and can be found in Appendix A.3. Raw results can be found in Appendix E.2.

6.1.1. Case Description

To properly test the benchmark and see if it is fit for use in a real-world scenario, a case study is designed. For this case study, the hypothetical business Gladiolas Gardening and Landscaping is used. Gladiolas Gardening was founded in 1998 by two sisters. They bought an old dilapidated post office and turned it into a wonderful garden center. Business has been blooming on that location ever since. From the period of 2010 till 2019 Gladiolas Gardening purchased and renovated another three dilapidated buildings with the help of government support. About a year ago, the sisters agreed upon facing a new challenge to expand their business even further. They contacted a financially struggling landscaping company called Larkspur and decided to merge. The two companies continued under the name Gladiolas Gardening and Landscaping, although the name Larkspur is still used to reference the landscaping industry.

Although the merge was a success and the service offered by the landscaping company is again profitable, the data management of both companies has become a struggle. Both companies have extensive databases with customer information and contain data on plants and flowers. However, after merging the two companies the amount of conflicting data turned out to be unworkable; customers associated to both businesses lost part of their loyalty points and plants for landscaping got delivered to old addresses. Additionally, Gladiolas Gardening and Landscaping wants to utilize the present knowledge on plants and flowers to improve their services. By adding knowledge on the lifespan of plants in different types of soil, they can provide their customers with better information on how to care for the purchased plants and flowers and provide a better estimation of the longevity of a plant. However, due to the rigidity of their current DBMS retrieving this type of data is not intuitively possible.

Gladiolas Gardening and Landscaping instructed their ICT-manager to explore the possibilities of probabilistic database technology. During an exploration on the internet, the ICT-manager stumbled upon the probabilistic database management systems DuBio and MayBMS, and found QuestionMark to evaluate the performance of both systems. As the company does now own extensive hardware, both systems are set up in a Docker container on a small server. The benchmark is also run on this hardware.

6.1.2. Benchmark Execution

For the execution of the benchmark, the steps provided in the manual of Appendix A and the steps in the manuals of both QuestionMark Python programs were precisely followed.

Phase 1: Reading through the manual and understanding the product
The first phase of the benchmarking process consists of getting to know the benchmark programs and to read through the manual. As the company does not have the resources to investigate the best possible probabilistic DBMS, it was decided to run the benchmark on both already included systems, which are MayBMS and DuBio.

Phase 2: Running QuestionMark: The Dataset Generator
The parameters of QuestionMark: The Dataset Generator were tweaked to represent the current dataset at the company as much as possible. The company is an SME (small to medium enterprise) with information from around 450 customers and around 1100 data points on plants and flowers. The dataset contains some uncertainty, but is not entirely uncertain. As the staff is already used to handling the present uncertain data in a deterministic manner, the new DBMS should not express doubt on every data point. Some incorrect information is preferred to processing overhead of probabilistic data. To represent this real-world dataset as closely as possible in the generated dataset, the following parameter values were chosen:

Variable	Value
dist	jaro
smaller_dataset	True
dataset_size	0.01
whole_clusters	False
non_bkv	[’description’, ’identifiers’, ’keyValuePairs’, ’price’, ’specTableContent’]
block	ASN
ws	2

phi	0.36
mbs	5
attributes	['brand', 'category', 'cluster_id', 'description', 'identifiers', 'keyValuePairs', 'price', 'specTableContent', 'title']
weights	[1, 0.7, 1, 0.8, 0.8, 0.8, 1, 0.7, 1]
lower_phi	0.28
upper_phi	0.36
dbms	MayBMS or DuBio
performance	False

Phase 3: Running QuestionMark: The Probabilistic Benchmark
Next, QuestionMark: The Probabilistic Benchmark was set up. For Gladiolas Gardening and Landscaping, select queries and insert statements are most often used. Since the company wants to have a complete overview, all queries are selected to run the benchmark on. For a clearer overview, the queries and statements were run in a separate benchmark process. Since a lightweight dataset is required, time will allow for a full benchmark run. Since the part of the database containing information on flowers and plants is often queried on the spot next to customers, relevant queries need to finish within five minutes. The following parameters are set in QuestionMark: The Probabilistic Benchmark:

Variable	**Value**
dbms	MayBMS or DuBio
iterations	5
timeout	300
show_query_plan	True
queries	The full list of queries

Phase 4: Digesting the results and drawing conclusions
During phase 4, the results generated by QuestionMark are digested by following the instructions from the manual in Appendix A. During the benchmark execution of both MayBMS and DuBio, several errors were thrown that had to be resolved to find out more information on the load handling capabilities of the selected DBMS. Therefore, the benchmarking process iterated several times over phases 3 and 4. The results produced during phase 4 are discussed in the next sections. For MayBMS, this is Section 6.1.3 and for DuBio Section 6.1.4.

6.1.3. Results MayBMS

This section provides an overview of the benchmark results of MayBMS. The raw results of the benchmark test for MayBMS can be found in Appendix E.2. The results of this benchmark test are obtained by following the instructions from the manual in Appendix A.5.

Effectiveness
QuestionMark could provide a somewhat clear image on the effectiveness of MayBMS. The metrics on effectiveness generated by QuestionMark can be found in Table 6.1. Due to compatibility issues between MayBMS and Psycopg2, statements that run with-

out error within a database tool crashed in QuestionMark. The percentage of successful queries is obtained by omitting the query parts that crash specifically with QuestionMark. From the benchmark test and additional error solving through a database tool, it can be concluded that MayBMS supports the functionalities as presented in Table 6.2.

Although MayBMS requires some attention due to its legacy code base, it was able to run almost all queries successfully. MayBMS cannot provide a compact representation of offers and their probabilities. Since MayBMS does not support logical disjuction within a single record. Therefore, when disjunction is present, multiple records need to be created for a single offer. MayBMS also does not support the most recent deterministic DBMS queries. As it is built into PostgreSQL 8.3.3., all functionality added to newer PostgreSQL versions is not supported.

Metric	**Value**
Percentage of successful queries	92.31%
Percentage of successful statements	100.0%

Table 6.1: Overview of all compact effectiveness metric values returned by MayBMS

#	Native	Possible	**Functionality**
1	[ ]	[ ]	Support of most recent deterministic DBMS queries
2	[ ]	[ ]	Offering a compact representation of the present uncertainty
3	[X]	[ ]	Get the probability of a an offer
4	[X]	[ ]	Get the probability of a composed result
5	[X]	[ ]	Apply aggregate functions on probabilities
6	[X]	[ ]	Filtering on probability
7	[X]	[ ]	Get the expected count
8	[X]	[ ]	Get the expected sum
9	[ ]	[X]	Get the most probable answer
10	[ ]	[X]	Verify if a specific possible world exists
11	[ ]	[X]	Verify if a record is certain
12	[ ]	[ ]	Updating the uncertainty of an offer
13	[ ]	[ ]	Repair the probability space after addition, update or deletion of offers
14			Any anomalies discovered during benchmarking

Table 6.2: Supported functionality by MayBMS

The following log was created while altering queries that have raised an exception:

Efficiency
QuestionMark can provide plenty information on the efficiency of MayBMS. The metrics on efficiency generated by QuestionMark can be found in Table 6.3. QuestionMark also generated graphs to provide insights into the character count and runtime of the queries. The graphs created on the character count can be found in Figure 6.2. The graphs on runtime can be found in Figure 6.1.

Fix log 1	
Query that raised an exception:	insight_2
Prior adaptations done on the query:	-
Exception raised by the query:	column ”offers._v0” must appear in the GROUP BY clause or be used in an aggregate function.
Suspected cause of the exception:	MayBMS does not know how to handle the probability space in a deterministic query.
Implemented fix:	Run the query on the setup table.

Fix log 2	
Query that raised an exception:	probabilistic_4
Prior adaptations done on the query:	Tried to find a solution to having the query also return the random variables that were used to calcuate the probabiltiy.
Exception raised by the query:	-
Suspected cause of the exception:	Lack of support.
Implemented fix:	Accepted it will not show this information, it will require an unreasonable large query.

Fix log 3	
Query that raised an exception:	All insert_update_delete
Prior adaptations done on the query:	-
Exception raised by the query:	column attrs. does not exist. syntax error at or near ”REPAIR”
Suspected cause of the exception:	MayBMS has legacy code that is not compatible with Psycopg2.
Implemented fix:	Tried the code in database tool and it worked. To obtain some information on the runtime, the query part that repairs the probability space is removed in all queries.

MayBMS can run the queries and statements given to it rather fast. MayBMS does not support fast changing datasets well. When data is added or uncertainty is updated, a new probability space should be created. This requires some additional queries. It should be noted that the displayed runtime of the statement does not include repairing the probability space. The actual runtime of statements is higher. MayBMS has a good character count for its queries, but is not very character efficient on its statements. For the queries, the query dialect hides the complexity of the underlying processes well, meaning that you can write queries succinctly with MayBMS. Finally, as MayBMS does not have a compact representation of the probability space, it uses over 80% of the dataset for duplicate data, random variables and probabilities. This makes MayBMS space inefficient.

Metric	**Value**
Total number of characters needed for all queries	1412 chars
Total number of characters needed for all statements	3626 chars
Percentage of successful queries	92.31%
Percentage of successful statements	100.0%
Total runtime of all queries	112.76 ms
Total runtime of all statements	111.05 ms
Total size of the probability space	3240 kB
Total size of duplicate records	875.5 kB
Percentage of data used for probabilistic representation	80.39%

Table 6.3: Overview of metric result returned by MayBMS

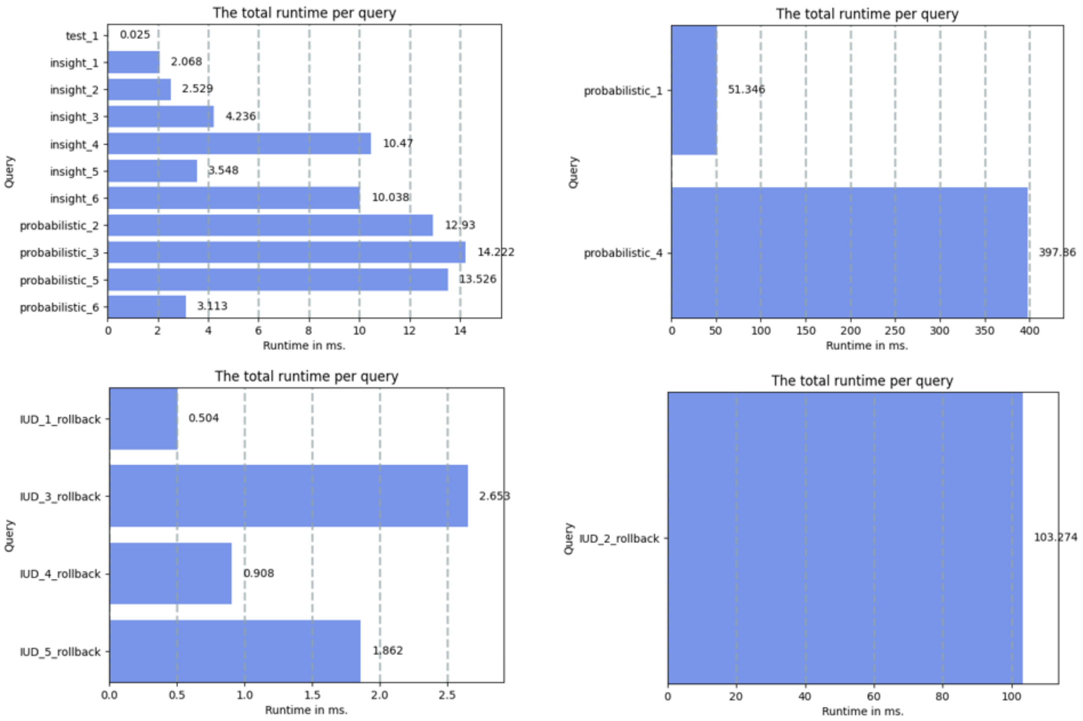


Figure 6.1: Runtime of all queries in MayBMS

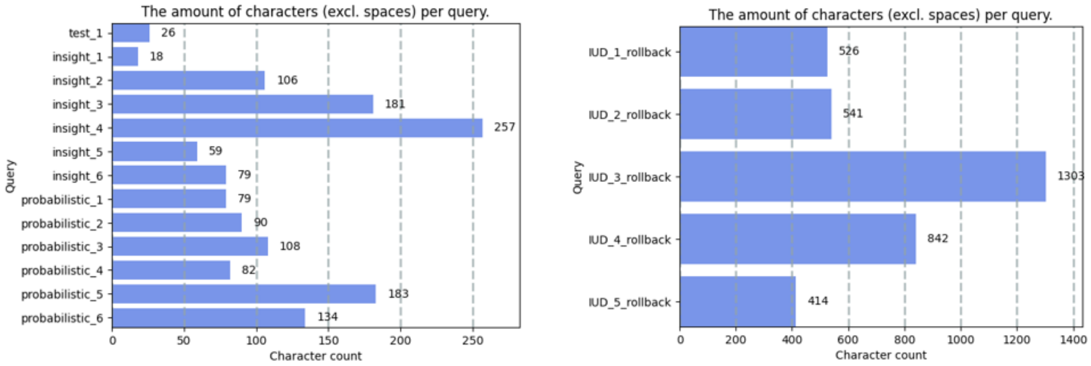


Figure 6.2: The character count for all queries in MayBMS

Appeal

MayBMS scores average on appeal. The software documentation is sufficient for basic queries, but once queries become more advanced documentation is lacking completely. It is, for example, not explained in literature how to repair the probability space once new records are included in a table. Recalculating this entirely each time new data is added is a lot of overhead for large tables. This also results that the software is not super easy to work with. Regarding expenses, MayBMS is completely free. Scoring MayBMS based on the appeal table in the user manual provides the result as presented in Table 6.4.

[1, 2, **3**, 4, 5]	The software is well documented.
[1, **2**, 3, 4, 5]	The software was easy to work with.
[1, 2, 3, **4**, 5]	We have sufficient in-house expertise to work well with the software.
[1, 2, 3, 4, **5**]	I am satisfied with the monetary expenses that need to be made for running the software.
[**1**, 2, 3, 4, 5]	The software has a support service.

Table 6.4: The appeal scores for MayBMS

6.1.4. Results DuBio

The raw results of the benchmark test for DuBio can also be found in Appendix E.2. Also for DuBio, the results of this benchmark test are obtained by following the instructions from the manual in Appendix A.5.

Effectiveness

QuestionMark could provide a clear image on the effectiveness of DuBio. The metrics on effectiveness generated by QuestionMark can be found in Table 6.5. From the benchmark test, it concludes that DuBio has a high effectiveness. DuBio supports the functionalities as presented in Table 6.6. A small x is used to indicate that it can only be run on small portions of the dataset, such as a specialized view. DuBio was able to run all insert, update an delete statements and could run most of the queries successfully.

DuBio is loosely built upon PostgreSQL, which ensures that all new functionalities of PostgreSQL are also supported in DuBio. It does not have native support for expected sum and expected count, but this can be calculated using aggregate functions. DuBio also does not have a good support for fast changing datasets. When data is added or uncertainty is updated, the user must manually update the probabilities in the dictionary. Additionally, DuBio struggles calculating the probability over large sentences and could only manage to do so for a dataset of 42 records in size.

Metric	**Value**
Percentage of successful queries	92.31%
Percentage of successful statements	100.0%

Table 6.5: Overview of all compact effectiveness metric values returned by DuBio

#	Native	Possible	**Functionality**
1	[X]	[ ]	Support of most recent deterministic DBMS queries
2	[X]	[ ]	Offering a compact representation of the present uncertainty
3	[X]	[ ]	Get the probability of an offer
4	[x]	[ ]	Get the probability of a composed result
5	[X]	[ ]	Apply aggregate functions on probabilities
6	[X]	[ ]	Filtering on probability
7	[ ]	[X]	Get the expected count
8	[ ]	[X]	Get the expected sum
9	[ ]	[X]	Get the most probable answer
10	[X]	[ ]	Verify if a specific possible world exists
11	[X]	[ ]	Verify if a record is certain
12	[ ]	[X]	Updating the uncertainty of an offer
13	[X]	[ ]	Repair the probability space after addition, update or deletion of offers
14			Any anomalies discovered during benchmarking

Table 6.6: Supported functionality by DuBio

The following log was created while altering queries that have raised an exception:

Fix log 1	
Query that raised an exception:	probabilistic_4
Prior adaptations done on the query:	-
Exception raised by the query:	The current running query has timed out.
Suspected cause of the exception:	DuBio requires too much time reading all probabilities from the dictionary.
Implemented fix:	Running the query on the part table.

Fix log 2	
Query that raised an exception:	probabilistic_4

Prior adaptations done on the query:	Running it on the part table
Exception raised by the query:	The current running query has timed out.
Suspected cause of the exception:	DuBio still requires too much time reading all probabilities from the dictionary.
Implemented fix:	Iteratively reducing the size of the part table until the query finishes.

Fix log 3	
Query that raised an exception:	probabilistic_4
Prior adaptations done on the query:	Reducing the size of the part table
Exception raised by the query:	Ran with dataset size of 42
Suspected cause of the exception:	-
Implemented fix:	Queried for a column timeout, so the benchmark run finishes.

Efficiency

QuestionMark provides sufficient information regarding the efficiency of DuBio. The metrics on efficiency generated by QuestionMark can be found in Table 6.7. The graphs created on the character count can be found in Figure 6.3. The graphs on runtime can be found in Figure 6.4.

The efficiency of DuBio is not optimal. When digesting a query calculating the probability over a large part of the dataset, DuBio starts to struggle and get a long execution time. In DuBio, each probability needs to be looked up in the dictionary table, which takes a significant amount of time. Also, the characters required to run all queries is rather long. DuBio currently does not hide its complexity, making that more characters are required. Also, the manual addition of probabilities to the dictionary in insert and update statements requires significant characters. DuBio does have a compact representation for the probabilistic data and is thus more space efficient.

Metric	**Value**
Total number of characters needed for all queries	1846 chars
Total number of characters needed for all statements	2138 chars
Total planning time of all queries	1.64 ms
Total execution time of all queries	472.02 ms
Total planning time of all statements	0.26 ms
Total execution time of all statements	10.63 ms
Total size of the _sentence column	160 kB
Total size of the dict table	784 kB
Percentage of data used for probabilistic representation	50.64%

Table 6.7: Overview of all compact efficiency metric values returned by DuBio

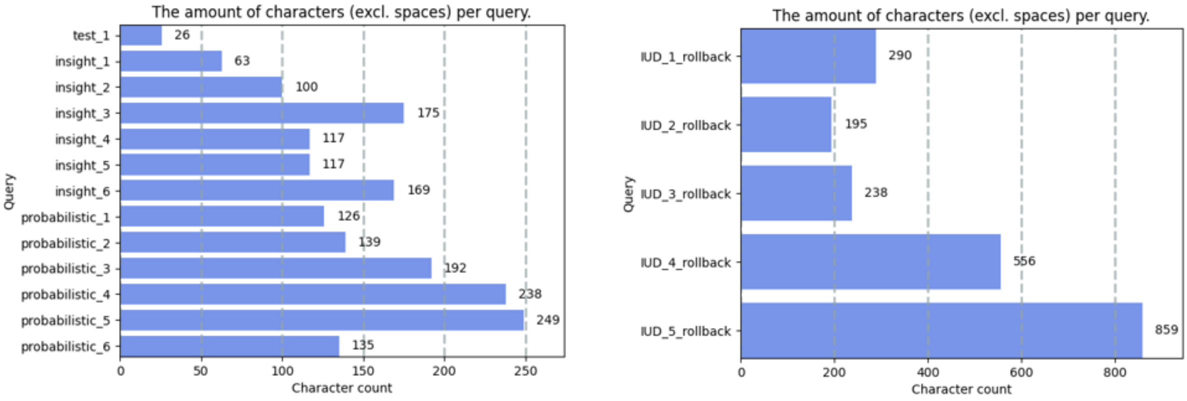


Figure 6.3: The character count for all queries in DuBio

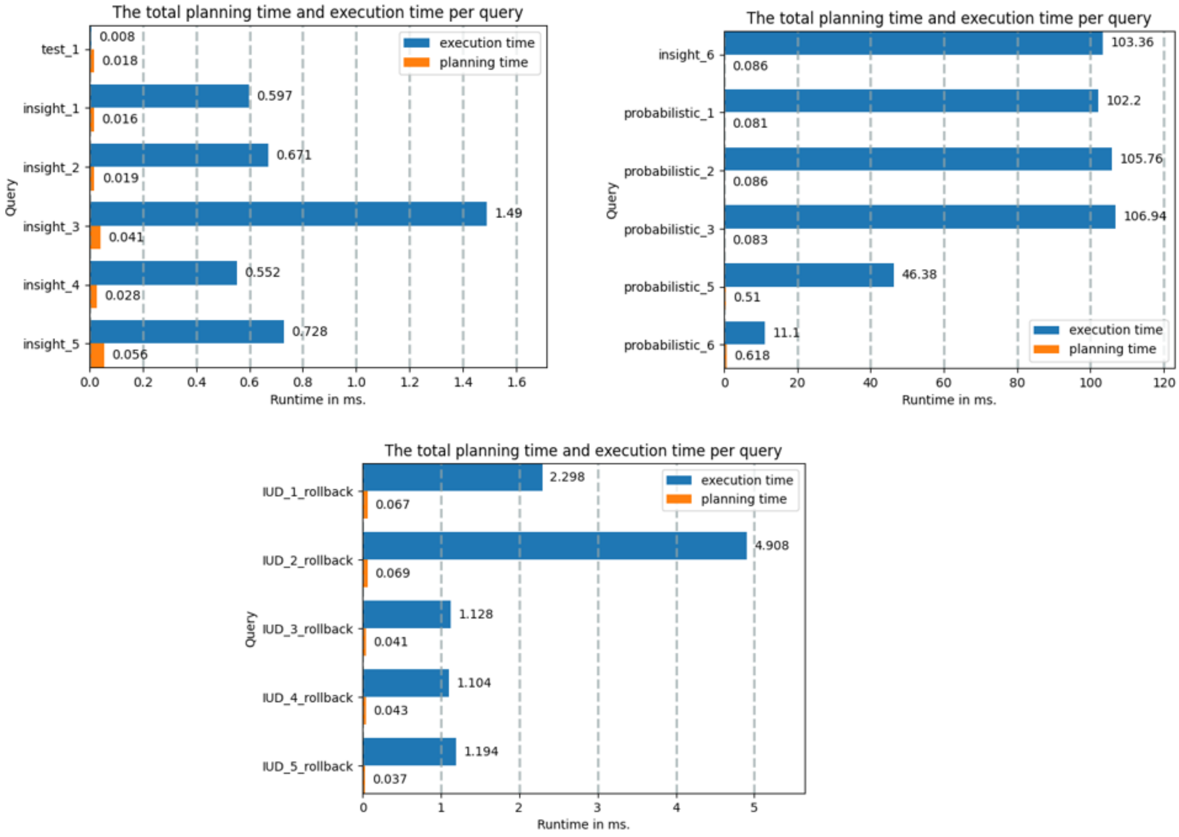


Figure 6.4: Runtime of all queries in DuBio

Appeal

The appeal of DuBio is also average. The software is currently not well documented, which makes it harder to work with. It is pleasant that it runs on the most recent PostgreSQL version. Also, DuBio is completely free. The appeal of DuBio is higher, as it is still under development. Aspects as documentation and ease of querying are thus likely to improve in the future. Scoring DuBio based on the appeal table in the user manual provides the result as presented in Table 6.8

[1, **2**, 3, 4, 5]	The software is well documented.
[1, 2, **3**, 4, 5]	The software was easy to work with.
[1, 2, 3, **4**, 5]	We have sufficient in-house expertise to work well with the software.
[1, 2, 3, 4, **5**]	I am satisfied with the monetary expenses that need to be made for running the software.
[**1**, 2, 3, 4, 5]	The software has a support service.

Table 6.8: The appeal scores for DuBio

6.1.5. Conclusion

The benchmark has successfully informed Gladiolas Gardening and Landscaping on the possibilities of both MayBMS and DuBio. Both systems still have their flaws, but their potential is great enough to benefit to the company. Although MayBMS seems to be able to process larger quantities of probabilistic data, DuBio comes with greater usability and promise, due to it being a more novel technology.

QuestionMark showed that it is capable of providing sufficient information regarding the effectiveness, efficiency and appeal of two different probabilistic database management systems. Although no unequivocal conclusions will be drawn in this paper regarding the best fitting probabilistic DBMS, as to ensure an unbiased view, it can be concluded that QuestionMark fulfills its purpose of informing its client of the capabilities of any tested DBMS.

6.2. User Testing

To verify the usability of the designed benchmark, user tests have been executed. The methodology used for this is described in Section 4.7. The exact instructions provided to the participants and the consent form can be found in Appendix D.

After each user test, the received feedback was implemented in the benchmark to improve its design. This ensured that new participants did not need to work through the same flaws. For each participant, a summary of their experimental results was produced. The extensive results can be found in Appendix D.4.

Participant #1
During the study, the laptop provided for the experiment posed some struggles to set up the Python environment required to run the study. The researcher guided the participant in setting up the environment to allow the participant to focus on the important part of the study. During the experiment is became clear that more explanation was required for several steps of the benchmarking process. Some phases of the benchmarking process also requires additional instructions by the researcher to inform the participant better on the imaginary situation they are participating in. During this study, it also became clear that the process of manually uncommenting and running the functions required for each step is a tedious process. The participant also indicated that too many clicks are required to navigate and use the system.

After the user study with participant #1, the following changes were implemented:

- The instructions in `MANUAL.md` of QuestionMark: The Dataset Generator were improved to provide clearer instructions on how to download the dataset, how to manually zip the produced files, and by making the explanation more concise by combining steps and removing instructions on the ISA algorithm.

- Additional instructions were provided during the benchmarking process, to indicate that the parameters are already correctly set and that a smaller dataset should be used.
- The process in `manual.py` was automated and the instructions in `MANUAL.md` were changed accordingly in both QuestionMark programs.
- The Python environment on the provided laptop was set up more robustly.
- `database.ini.tmpl` was updated to also contain a port field.

Participant #2
The laptop provided for the experiment again posed some struggles with the Python environment. Again, the researcher guided the participant in setting up the environment. This participant also indicated that too many clicks are required for navigating the systems. The subject indicated that the manual is clear and elaborate. More explanation on how and why of probabilistic benchmarking and setting up a database connection is desired.

After the user study with participant #2, the following changes were implemented:

- A Python environment with the project already downloaded is setup for the participants. They will be directed to the already set up environment once they begin the process of downloading the software.
- The manual of both QuestionMark systems are improved. It is indicated more clearly that the performance tests are optional. The overall feel of both manuals are made more similar.

Participant #3
This participant indicated that the manual provided was extensive and clear, but also noted that additional information regarding the included processes and the importance of benchmarking is desired. Again, it was indicated that the system requires too many clicks before it can be used. The participant commented that they were very much able to follow the instructions provided by the manual, but that they did not have an understanding of what they were doing. The provided results of the benchmark are clearly displayed, but it is still hard to understand what the provided results mean. Additional information is required and a graphical display is desired. The participant also indicated that they would like to see a ten minute video on how to use the benchmark. Several other comments were also made.

After the user study with participant #3, the following changes were implemented:

- Both manuals were updated to include the provided feedback.
- Typos in the software and manual were removed.
- More explanation on the benchmark queries was included.

Participant #4
This participant did not struggle much during the benchmarking process. They indicated that the manual is clear and that the software was easy to use. The results provided by the benchmark should get some additional information, as it was not yet clear what the provided results acutally meant.

After the user study with participant #4, the following changes were implemented:

- The performance results of QuesionMark: The Dataset Generator are enhanced with graphs.
- The benchmark results of QuestionMark: The Probabilistic Benchmark are enhanced with graphs and additional explanation.
- Additional explanation was added to the manuals using drop-down menus.

Participant #5
During the study the participant did not struggle much with running the benchmark. The participant indicated that they wanted to have more information on the internal processes of the benchmark, but it was observed that the participant skipped the parts containing additional explanation. The participant appreciated the way the program was build and indicated that it was easy to use.

After the user study with participant #5, the following changes were implemented:

- The user manual was updated and minor error were removed. Another instruction is added to indicate where the user can find additional information.
- The extensive user manual, which is also included as Appendix A in this thesis, was added to both projects.
- The directory structure in both projects was changed to hide files that the user does not need to interact with.

Participant #6
This participant also did not struggle much with running the benchmark. The participant noticed some inconsistencies within the naming of files, which required some additional instructions to clarify the new name or location. The main feedback provided by the participant, is that they wanted to know what the requirements are for running the benchmark. What Python version is required? Are there external dependencies in the project? The participant provided instructions on how to easily include this. The participant als indicated that the profided results should also be machine readable. A JSON file should be created. This also allows for a quicker human overview of the results.

After the user study with participant #6, the following changes were implemented:

- The manual was updated to include the correct file names and locations, break down some steps and clarify the step regarding the database connection.
- The required specifications are included in both programs and the manual.
- The finish message in TEST is altered to only display when there are actually no errors occurring.

- The errors thrown during the benchmarking process are not displayed within the run script anymore.

7 Conclusion

In this chapter, the results will be discussed and the findings of this research will be summarised. Future work will also be identified and a reflection upon the limitations of this research will be given.

7.1. Discussion

This research was performed to obtain an answer on two main research questions. The first research question sought an answer to how a benchmark can be designed to test and compare probabilistic database management systems on real-world strain. The second sought an answer to how the novel probabilistic database DuBio and the state-of-the-art MayBMS perform when benchmarking these technologies with the developed benchmark. In this research, both questions have successfully been answered.

For answering the first research question, focus was put on literature study. With the literature study, a theoretical framework was defined, which the design of QuestionMark could follow. The existing literature regarding benchmarking is extensive, although this is mostly limited to business processes. Regarding database benchmarking specific, fewer literature could be found. Literature on how to build a benchmark for probabilistic databases was lacking. QuestionMark was thus built upon the pillars established by literature on the benchmarking of deterministic databases and benchmarking in general. This paper contributes to the research on the design of benchmarking systems for probabilistic databases specifically.

Following the theoretical foundations, the QuestionMark benchmark for probabilistic databases was built. The main idea behind QuestionMark is that it is real-world approaching. By offering a benchmark that approaches the real-world, systems can be evaluated based on strain that they are likely to encounter once they get deployed. This research is thus the next step towards the widespread use of probabilistic databases outside the academic world.

This research also went beyond the creation of QuestionMark and put it to the test. This part answered the second research question. This testing was done by implementing support for two probabilistic database systems: MayBMS and DuBio. By implementing the novel system DuBio and the state-of-the-art MayBMS, the performance and usability of the benchmark could be verified. Apart from generating useful feedback on QuestionMark, it also provided insights into the performance of both database systems. The implementation of these system contributed significantly to the improvement of QuestionMark. The user testing performed with QuestionMark also led to useful insights, improving on the design even further. Both from the case study performed and from the feedback collected during the user study, it can be concluded that QuestionMark is a user friendly system that fulfills its purpose of benchmarking systems. QuestionMark can provide sufficient information on key areas of a system and can guide the user in gaining meaningful insight.

The design and development of QuestionMark enables the evaluation and comparison of the performance of a wide range of probabilistic database management systems. QuestionMark is another step towards the widespread use of probabilistic databases. The design of QuestionMark presented in this research is robust and capable of providing a large range of metrics determining the performance of both DuBio and MayBMS. Due to the open source nature of QuestionMark, support for other systems can be implemented when desired. The extensive manual guides the user in the use of the system.

7.2. Limitations

During this research, some limitations were encountered. This section will describe the limitations and provide possible solutions for them. The limitations are listed in alphabetical order.

Benchmark design flaws. The design of the benchmark codebase does have its limitations. Its major limitation is that the code is based upon PostgreSQL based database systems. Therefore, whenever a non-PostgreSQL based system should be benchmarked, significant alterations to the database connection code should be made. Another, similar, limitation is that someone with programming experience is needed even when a PostgreSQL based DBMS needs to be benchmarked. This limitation is hard to bypass, as each probabilistic DBMS has its own manner of processing uncertainty, which should be programmed into the code. Also, the benchmark does not yet have a graphical user interface. Currently the benchmark needs to be run from the code itself, thus also requiring someone with minor programming experience. What also poses problems with running statements with MayBMS, is the collaboration with the Psycopg2 Python library. Both technologies are nearing to legacy and thus show some strange behaviour when collaborating. This causes MayBMS statements to raise exceptions, that should be able to run fine. Finally, as the code has not been tested exhaustively, bugs can still be present in the code.

Dataset generation flaws. The Dataset Generator is also not flaw-free. Firstly, the generation of the probabilistic data is based on the possible worlds theory. This possible worlds theory is what DuBio and MayBMS also use to represent the uncertain data. It could be that the generation of the dataset following this method favours database management systems that are also based upon this representation. As the possible worlds generation was implemented mainly following a DuBio research paper, this could influence the fairness. Nevertheless, it is deemed unlikely that the way the dataset is generated influences the performance.

As a second, more prominent, flaw, The Dataset Generator has a limitation on the maximum cluster size it can generate. The current implementation of the generation of possible worlds imposes a limitation on the maximum cluster size that can be generated. The generation of possible worlds from a selection of offers grows factorially, meaning that the calculation time needed explodes when generating a probabilistic cluster from a block of seven offers.

A third flaw is that the performance of The Dataset Generator is lacking. As can be seen in Appendix C, the performance of both the blocking algorithms as the matching algorithm is falling short. This results in a dataset generation that is less credible as a real-world scenario. As no extensive research was conducted to include the best blocking and matching algorithm, the performance could also lack due to a suboptimal algorithm choice. It should be noted that a precision and recall of 100% is not desired, as there would then be no uncertainty, but the current precision of 46% and recall of 33% are deemed too low. It should be noted that the performance of the benchmark is not influenced by the quality of the dataset.

Dataset limitations. As mentioned in Section 4.2, the dataset was selected based on a set of defined requirements. While working with the dataset, limitations on the chosen dataset were discovered. First, the dataset has most of its data in one big table, making that there are limited possibilities with join-queries. Second, there are no data attributes on which numeric operations can be performed. Sum-queries are now performed on the cluster ID, which is not a real-world application of that column. To omit these limitations, effort can be put into finding a dataset that is more fit or to adapt this dataset to be more generalisable.

More familiar with DuBio than MayBMS. Although best efforts were made to design the benchmark as independently from DuBio and MayBMS as possible, previous knowledge on both systems cannot be ignored. The possibilities of both systems were extensively studied during the definition of the queries to eliminate a skew of expertise. Nevertheless, it cannot be determined for certain if a favour bias towards DuBio is fully absent. An external study should confirm the validity of the created benchmark.

Hardware limitations. During this research, it was chosen to develop and test QuestionMark on the same small machine. Due to the limited capabilities of that machine, the benchmark has only been run with the smallest dataset possible and with limited uncertainty in the generated dataset. Although it is assumed that the size of the dataset does not impact the performance of the benchmark itself, this is not explicitly verified.

Threats to Fairness. In Section 3.3 eleven aspects to avoid to obtain a fair benchmark were presented. From these eleven, it was actively decided not to perform the benchmark with a cold start, as suggested for a fair impression by Hohenstein and Jergler [41]. This decision was made since clearing a cache fully requires significant effort and with the current setup it cannot be verified if a cold start is truly cold. Additionally, running queries in a real-world business setting is almost never truly cold. There is often some relevant data in the cache that results in a lukewarm start of a query. The benchmark thus runs queries with a warm start.

Another aspect from [41] that might influence the fairness is any incorrectness in the code. Best efforts were made to include high quality code. By having the codebase open source, correctness of the code can be validated by third parties. As the code is not too complex and the system is user tested, it is assumed that no incorrectness in the code exists to skew the performance of any system.

As an additional threat to fairness, prior experience with DuBio cannot be ignored. Although MayBMS and other probabilistic database systems were studied extensively before designing the benchmark, it must be acknowledged that DuBio remained the most familiar system. This imbalance in experience with the systems was acknowledged throughout the design of the benchmark and best efforts were made to compensate for it.

As also suggested by Hohenstein and Jergler, the database should be tuned for each probabilistic DBMS. As this requires expertise and takes much effort to get it right, this step was omitted. It is not verified whether the performance of any DBMS is significantly harmed by the default settings.

Query set flaws. For this benchmark, best efforts were made to obtain a set of queries that is as complete and representative as possible. A literature review and case study were conducted to obtain a diverse set of queries. However, due to limited prior experience with querying databases the query set might not be fully real-world representative. Peer review should validate the query set and additions or alterations to this set could be made.

7.3. Future work

Based on the results of this research, several directions for future research can be identified. As the main direction of future research, the benchmark presented in this research should be thoroughly tested to improve on its design even further. Although this research showed promising results from the benchmark, it was only tested in a controlled environment. Further testing of the benchmark should indicate whether the benchmark is user friendly and whether it contains all required functionality. Although this research is a good first step, the benchmark is most likely not yet perfect. More research should show if specific probabilistic database functionalities are not yet sufficiently covered and if the benchmark is truly fair.

This benchmark can also be used to evaluate the performance of several probabilistic database management systems. Due to the broad functionality coverage by QuestionMark, systems can be put to the test thoroughly. By running the benchmark on a machine with better specifications, larger clusters can be formed, putting the probabilistic systems under higher pressure. Also, QuestionMark can be updated to support additional probabilistic database management systems. Feedback collected from that process can further improve the design of QuestionMark and make it an even better system.

Regarding QuestionMark: The Dataset Generator, future work should focus on improving the performance of the blocking and matching algorithms. Although the efficiency and functionality support of a probabilistic database management system is not influenced by the quality of the dataset, query results cannot be used to gain useful information from. Ideally, the precision and recall of the dataset can be tweaked to obtain a dataset ranging from very low to high uncertainty.

Another issue worth addressing is the generation of the possible worlds in QuestionMark: The Dataset Generator. Due to the theoretical approach taken, the calculation of probabilistic clusters explodes when their original cluster contains seven or more products. It should be researched whether a more resource-efficient approach can be implemented.

Since probabilistic databases are not yet widely used outside of the academic world, this benchmark can also aid in showing the added value that the use of probabilistic database systems can have. For this, support for deterministic PostgreSQL queries can be implemented. This way, the results provided by a deterministic approach versus a probabilistic approach can be compared. Running the benchmark on PostgreSQL could also aid in providing a baseline measure to compare the efficiency of probabilistic systems with. With this, the trade-off between the higher execution time and high-value information can be visualised.

7.4. Conclusion

In this research, The QuestionMark Probabilistic Benchmark is presented. QuestionMark is a benchmark to obtain information on the effectiveness, efficiency and appeal of any probabilistic database management system. The benchmark is designed to cover a wide range of functionalities, so that any application area can be tested. As the benchmark is real-world approaching it can provide a true image of the performance of the tested system.

The aim of this research was to design a benchmark that can be used to test probabilistic database management systems on real-world strain. The design cycle as presented in this research resulted in the development of the QuestionMark benchmark for probabilistic databases. QuestionMark consists of two Python programs that together run the benchmark. User testing has shown that QuestionMark is easy in use, although minor knowledge on python is required. Due to the extensive manual and its open source nature, QuestionMark can easily be extended to support various systems.

The QuestionMark benchmark for probabilistic databases is another step in the direction of widespread use of probabilistic database management systems outside of the academic world. With this benchmark, developers of probabilistic database systems can easily strain test their software and improve their product. Likewise, consumers of probabilistic database technology can use QuestionMark to find what software fits them best. QuestionMark is ready to guide the future of databases.

References

[1] L. Akritidis and P. Bozanis. “Effective Unsupervised Matching of Product Titles with k-Combinations and Permutations”. In: *2018 IEEE (SMC) International Conference on Innovations in Intelligent Systems and Applications* (2018).

[2] L. Akritidis et al. *A self-verifying clustering approach to unsupervised matching of product titles*. Vol. 53. 7. Springer Nature B.V., 2020, pp. 4777–4820.

[3] A. Ali, S. Talpur, and S. Narejo. “Detecting Faulty Sensors by Analyzing the Uncertain Data Using Probabilistic Database”. In: *2020 3rd International Conference on Computing, Mathematics and Engineering Technologies: Idea to Innovation for Building the Knowledge Economy* (2020), pp. 3–9.

[4] G. Anand and R. Kodali. “Benchmarking the benchmarking models”. In: *Benchmarking* 15.3 (2008), pp. 257–291.

[5] L. Antova, C. Koch, and D. Olteanu. “$10^{(10^6)}$ worlds and beyond: Efficient representation and processing of incomplete information”. In: *VLDB Journal* 18.5 (2009), pp. 1021–1040.

[6] L. Antova, C. Koch, and D. Olteanu. *MayBMS*. 2008. URL: `http://maybms.sourceforge.net/` (visited on 05/03/2022).

[7] L. Antova, C. Koch, and D. Olteanu. “MayBMS: A Possible Worlds Base Management System”. In: (2006).

[8] L. Antova, C. Koch, and D. Olteanu. “MayBMS: Managing incomplete information with probabilistic world-set decompositions”. In: *Proceedings - International Conference on Data Engineering* (2007), pp. 1479–1480.

[9] L. Antova et al. “Fast and Simple Relational Processing of Uncertain Data”. In: *IEEE 24th International Conference on Data Engineering* (2008), pp. 983–992.

[10] D. Ayala et al. “Multi-source dataset of e-commerce products with attributes for property matching”. In: *Data in Brief* 41 (2022), pp. 1–6.

[11] N. Ayat et al. “Entity resolution for probabilistic data”. In: *Information Sciences* 277 (2014), pp. 492–511.

[12] C. Back and D. L. Scapin. “Comparing Inspections and User Testing for the Evaluation of Virtual Environments”. In: *International Journal of Human-Computer Interaction* 26 (8 2010).

[13] M. Balazinska et al. “Data management in the worldwide sensor web”. In: *IEEE Pervasive Computing* 6.2 (2007), pp. 30–40.

[14] I. Bhattacharya and L. Getoor. “Collective entity resolution in relational data”. In: *ACM Transactions on Knowledge Discovery from Data* 1 (1 2007).

[15] K. Booijink. “Evaluating the Scalability of MayBMS, a Probabilistic Database Tool”. Bachelor’s Thesis. University of Twente, 2019.

[16] L. E. Budde et al. “Development of a Database for Benchmark Datasets in Photogrammetry and Remote Sensing”. In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* V-1-2022.June (2022), pp. 187–193.

[17] K. Cao and H. Liu. “Entity Resolution Algorithm for Heterogeneous Data Sources”. In: *Proceedings - 2021 International Conference on Computer Information Science and Artificial Intelligence, CISAI 2021* (2021), pp. 553–557.

[18] I. I. Ceylan, A. Darwiche, and G. van Den Broeck. “Open-world probabilistic databases: An abridged report”. In: *IJCAI International Joint Conference on Artificial Intelligence* (2017), pp. 4796–4800.

[19] I. Ilkan Ceylan, A. Darwiche, and G. van Den Broeck. “Open-world probabilistic databases”. In: *Proceedings of the International Conference on Knowledge Representation and Reasoning* (2016), pp. 339–348.

[20] V. Cincotta. “Design and Implementation of a Scalable Probabilistic Database System”. MA thesis. Università di Genova, 2019.

[21] W. Dai and D. Berleant. “Benchmarking Contemporary Deep Learning Hardware and Frameworks: a Survey of Qualitative Metrics”. In: *019 IEEE 1st International Conference on Cognitive Machine Intelligence, CogMI 2019* (2019), pp. 148–155.

[22] N. Dalvi, C. Ré, and D. Suciu. “Probabilistic databases: Diamonds in the dirt”. In: *Communications of the ACM* 52.7 (2009), pp. 86–94.

[23] N. Dalvi and D. Suciu. “Management of Probabilistic Data: Foundations and Challenges”. In: *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (2007), pp. 1–12.

[24] R. Dattakumar and R. Jagadeesh. “A review of literature on benchmarking”. In: *Benchmarking: An International Journal* 10.3 (2003), pp. 176–209.

[25] T. De Vries et al. “Robust record linkage blocking using suffix arrays and bloom filters”. In: *ACM Transactions on Knowledge Discovery from Data* 5.2 (2011), pp. 1–27.

[26] D. Dominguez-Sal, N. Martinez-bazan, and V. Muntes-mulero. “Performance Evaluation, Measurement and Characterization of Complex Systems - Second TPC Technology Conference, TPCTC 2010, Revised Selected Papers”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6417 LNCS (2011), pp. 25–40.

[27] M. Dylla, I. Miliaraki, and M. Theobald. “A temporal-probabilistic database model for information extraction”. In: *Proceedings of the VLDB Endowment* 6.14 (2013), pp. 1810–1821.

[28] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. “Duplicate Record Detection: A Survey”. In: *IEEE Transactions on Knowledge and Data Engineering* 19 (1 2007).

[29] V. Ercegovac, David J. Dewitt, and R. Ramakrishnan. “The TEXTURE benchmark: Measuring performance of text queries on a relational DBMS”. In: *VLDB 2005 - Proceedings of 31st International Conference on Very Large Data Bases* 1 (2005), pp. 313–324.

[30] J. Flokstra, M. van Keulen, and Nikki Zandbergen. *wdc data converter*. 2022. URL: `https://github.com/utwente-dmb/wdc_pdb` (visited on 06/22/2022).

[31] P. V. Freytag and S. Hollensen. “The process of benchmarking, benchlearning and benchaction”. In: *The TQM Magazine* 13.1 (2001), pp. 25–33.

[32] L. Fu, G. Salvendy, and L. Turley. “Effectiveness of user testing and heuristic evaluation as a function of performance classifcation”. In: *Behaviour and Infomration Technology* 21 (2 2002).

[33] C. J. Gillan et al. “Expediting assessments of database performance for streams of respiratory parameters”. In: *Computers in Biology and Medicine* 100.May (2018), pp. 186–195.

[34] G. Grahne. “Dependency Satisfaction In Databases With Incomplete Information”. In: *Proceedings of the Tenth International Conference on Very Large Data Bases* (1984), pp. 37–45.

[35] J. Gray. “The Benchmark Handbook for Database and Transaction Systems”. In: *The Benchmark Handbook for Database and Transaction Systems* (1993), pp. 1–15. URL: `http://research.microsoft.com/en-us/um/people/gray/benchmarkhandbook/chapter1.pdf`.

[36] S. Gregor and A. R. Hevner. “Positioning and presenting design science research for maximum impact”. In: *MIS Quarterly* 37.2 (2013), pp. 337–355.

[37] J. Groot Roessink. “inSQeLto: a Query Language for Probabilistic Databases”. Bachelor’s Thesis. 2021.

[38] M. Hassenzahl, S. Diefenbach, and A. Goritz. “Needs, affect, and interactive products – Facets of user experience”. In: *Interacting with Computers* 22 (2010).

[39] M. Hassenzahl et al. “Hedonic and Ergonomic Quality Aspects Determine a Software’s Appeal”. In: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (2000).

[40] T. Hirsch and B. Hofer. “A Systematic Literature Review on Benchmarks for Evaluating Debugging Approaches”. In: *The Journal of Systems & Software* 192 (2022), pp. 1–17.

[41] U. Hohenstein and M. Jergler. *About the fairness of database performance comparisons*. Vol. 1255 CCIS. Springer International Publishing, 2020, pp. 136–156.

[42] T. Imielinski and L. Witold Jr. “Incomplete Information And Dependencies In Relational Databases”. In: *Proceedings of the 1983 ACM SIGMOD internation conference on Management of data* (1983), pp. 178–184.

[43] R. Jampani et al. “MCDB: A monte carlo approach to managing uncertain data”. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data* (2008), pp. 687–700.

[44] A. S. Jumde and N. S. Chaudhari. “Query processing techniques in probabilistic databases”. In: (2016), pp. 483–488.

[45] Kaggle Inc. *Kaggle*. 2010. URL: `https://www.kaggle.com/datasets` (visited on 05/23/2022).

[46] C. Koch. “MayBMS: A System for Managing Large Uncertain and Probabilistic Databases”. In: (2009), pp. 1–34.

[47] C. Koch and D. Olteanu. “Conditioning probabilistic databases”. In: *Proceedings of the VLDB Endowment* 1.1 (2008), pp. 313–325.

[48] C. Koch et al. *MayBMS: A Probabilistic Database System*. 2009.

[49] H. Köpcke et al. “Tailoring entity resolution for matching product offers”. In: *ACM International Conference Proceeding Series* (2012), pp. 545–550.

[50] L. V. S. Lakshmanan, R. Ross, and V. S. Subrahmanian. “ProbView: A Flexible Probabilistic Database System”. In: *ACM Transactions on Database Systems* 22.3 (1997), pp. 419–469.

[51] W. M. Lankford. “Benchmarking: Understanding the Basics”. In: *The Coastal Business Journal* 1.1 (2002).

[52] J. Li et al. “Deep cross-platform product matching in e-commerce”. In: *Information Retrieval Journal* 23.2 (2020), pp. 136–158.

[53] R. Likert. “A technique for the measurement of attitudes.” In: *Archives of psychology.* (1932).

[54] A. Makris et al. “MongoDB Vs PostgreSQL: a comparative study on performance aspects”. In: *GeoInformatica* 25.1 (2021), pp. 241–242.

[55] R. R. Mauritz et al. “A probabilistic database approach to autoencoder-based cleaning”. In: *ACM Journal of Data and Information Quality, Special Issue on Deep Learning for Data Quality* (Jan. 2021). eprint: `2106.09764`.

[56] P. H. Meade. “A Guide to Benchmarking”. In: (2007), pp. 4–22.

[57] B. Mozafari et al. “Scaling up crowdsourcing to very large datasets: A case for active learning”. In: *Proceedings of the VLDB Endowment* 8.2 (2014), pp. 125–136.

[58] S. Mudgal et al. “Deep learning for entity matching: A design space exploration”. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data* (2018), pp. 19–34.

[59] R. B. Myers and J. R. Herskovic. “Probabilistic techniques for obtaining accurate patient counts in Clinical Data Warehouses”. In: *Journal of Biomedical Informatics* 44.SUPPL. 1 (2011), pp. 69–77.

[60] J. Nielsen. *Heuristic Evaluation*. John Wiley Sons, Inc., 1994, pp. 25–62.

[61] Oxford Languages. *Oxford Dictionary of English*. 3rd ed. Oxford University Press, 2010.

[62] F. Panse, M. van Keulen, and N. Ritter. “Indeterministic handling of uncertain decisions in deduplication”. In: *Journal of Data and Information Quality* 4.2 (2013).

[63] F. Panse et al. “Duplicate detection in probabilistic data”. In: *Proceedings - International Conference on Data Engineering* (2010), pp. 179–182.

[64] G. Papadakis et al. “Blocking and Filtering Techniques for Entity Resolution: A Survey”. In: *ACM Computing Surveys* 53.2 (2020).

[65] R. Peeters et al. “Using schema.org Annotations for Training and Maintaining Product Matchers”. In: *ACM International Conference Proceeding Series* Part F1625 (2020), pp. 195–204.

[66] K. Peffers, T. Tuunanen, and M. A. Rothenberger. “A design science research methodology for information systems research”. In: *Journal of Management Information Systems* 24.3 (2007), pp. 45–77.

[67] A. Primpeli, R. Peeters, and C. Bizer. “The WDC training dataset and gold standard for large-scale product matching”. In: *The Web Conference 2019 - Companion of the World Wide Web Conference, WWW 2019* (2019), pp. 381–386.

[68] M. Raasveldt et al. “Fair benchmarking considered difficult: Common pitfalls in database performance testing”. In: *Proceedings of the Workshop on Testing Database Systems, DBTest 2018* (2018).

[69] S. Ray, B. Simion, and A. D. Brown. “Jackpine: A benchmark to evaluate spatial database performance”. In: *Proceedings - International Conference on Data Engineering* (2011), pp. 1139–1150.

[70] C. Ré and D. Suciu. “Management of Data with Uncertainties”. In: *16th ACM Conference on Information and Knowledge Management* (2007), pp. 3–7.

[71] K. van Rijn. “A Binary Decision Diagram based approach on improving Probabilistic Databases”. Bachelor’s Thesis. University of Twente, 2020.

[72] P. Ristoski et al. “A machine learning approach for product matching and categorization”. In: *Semantic Web* 9.5 (2018), pp. 707–728.

[73] J. Schoenfisch and H. Stuckenschmidt. “Analyzing real-world SPARQL queries and ontology-based data access in the context of probabilistic data”. In: *International Journal of Approximate Reasoning* 90 (2017), pp. 374–388.

[74] P. Sen, A. Deshpande, and L. Getoor. “PrDB: Managing and exploiting rich correlations in probabilistic databases”. In: *The VLDB Journal* 18.5 (2009), pp. 1065–1090.

[75] S. M. A. Shah et al. “Robustness Testing of Embedded Software Systems: An Industrial Interview Study”. In: *IEEE Access* 4 (2016).

[76] A. Souihli and P. Senellart. “Optimizing approximations of DNF query lineage in probabilistic XML”. In: *Proceedings - International Conference on Data Engineering* (2013), pp. 721–732.

[77] Statista. *Ranking of the most popular relational database management systems worldwide, as of January 2022*. 2022. URL: `https://www.statista.com/statistics/1131568/worldwide-popularity-ranking-relational-database-management-systems/` (visited on 05/06/2022).

[78] M. Stonebraker and L. A. Rowe. “The Design of POSTGRES”. In: *ACM SIGMOD Record* 15.2 (1986), pp. 340–355.

[79] P. E. Strandberg et al. “Instrument from: Test Results Communication – An Interview Study in the Embedded Software Industry”. In: (2018).

[80] J. C. Strauss. “A benchmark study”. In: *Fall Joint Computer Conference*. 1972, pp. 1225–1233.

[81] S. S. Tee, T. S. M. T. Wook, and S. Zainudin. “User Testing for Moodle Application”. In: *International Journal of Software Engineering and its Applications* 7 (5 2013).

[82] The PostgreSQL Global Development Group. *PostgreSQL*. 1996. URL: `https://www.postgresql.org/` (visited on 05/06/2022).

[83] The PostgreSQL Global Development Group. *PostgreSQL 14.2 Documentation*. 2022.

[84] University of Twente. *Ethics Committee Computer & Information Systems*. 2023. URL: `https://www.utwente.nl/en/eemcs/research/ethics/` (visited on 04/14/2023).

[85] M. Q. T. P. van der Arend, J. F. Beerten, and M. van Keulen. “Benchmarking MayBMS based on hardware specifications and query complexity”. In: 2020.

[86] M. van Keulen. “Probabilistic Data Integration”. In: *Encyclopedia of Big Data Technologies* (2019), pp. 1308–1315.

[87] M. van Keulen and A. De Keijzer. “Qualitative effects of knowledge rules and user feedback in probabilistic data integration”. In: *VLDB Journal* 18.5 (2009), pp. 1191–1217.

[88] M. van Keulen and J. Flokstra. *DuBio*. 2019. URL: `https://github.com/utwente-db/DuBio` (visited on 05/03/2022).

[89] M. van Keulen et al. *Rule-based conditioning of probabilistic data*. Vol. 11142 LNAI. Springer International Publishing, 2018, pp. 290–305.

[90] B. Wanders, M. van Keulen, and P. van der Vet. “Uncertain groupings: Probabilistic combination of grouping data”. In: *Lecture Notes in Computer Science* 9261.September 2015 (2015), pp. 236–250.

[91] B. Wanders and M. van Keulen. “Revisiting the formal foundation of Probabilistic Databases”. In: *Proceedings of the 2015 Conference of the International Fuzzy Systems Association and the European Society for Fuzzy Logic and Technology* 89 (2015).

[92] B. Wanders, M. van Keulen, and P. van der Vet. “Uncertain groupings: Probabilistic combination of grouping data”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9261 (2015), pp. 236–250.

[93] Web Data Commons. *Training Dataset and Gold Standard for Large-Scale Product Matching*. 2018. URL: `http://webdatacommons.org/largescaleproductcorpus/v2/` (visited on 05/17/2022).

[94] J. Widom. “Trio: A system for integrated management of data, accuracy, and lineage”. In: *2nd Biennial Conference on Innovative Data Systems Research* (2005), pp. 262–276.

[95] R. J. Wieringa. *Design science methodology*. 2014, pp. 1–332.

[96] M. Wilke and E. Rahm. “Towards Multi-Modal Entity Resolution for Product Matching”. In: *CEUR Workshop Proceedings* 3075 (2021).

[97] S. Yan et al. “Adaptive sorted neighborhood methods for efficient record linkage”. In: *Proceedings of the ACM International Conference on Digital Libraries* (2007), pp. 185–194.

[98] N. Zandbergen. *QuestionMark: The Dataset Generator*. 2022. URL: `https://github.com/utwente-db/QuestionMark/tree/thedatasetgenerator` (visited on 06/25/2023).

[99] N. Zandbergen. *QuestionMark: The Probabilistic Benchmark*. 2022. URL: `https://github.com/utwente-db/QuestionMark/tree/theprobabilisticbenchmark` (visited on 06/25/2023).

[100] Z. Zhang et al. “MWPD2020: Semantic web challenge on mining the web of html-embedded product data”. In: *CEUR Workshop Proceedings* 2720 (2020).

A

The QuestionMark Probabilistic Benchmark

Note: this appendix contains the product manual provided with the QuestionMark software. As this manual is developed as a standalone product, the content in this appendix partially overlaps with the content from the thesis.

Welcome to the QuestionMark manual. This document provides additional details on the benchmark, as well as a roadmap on how to use it and adapt it for own use. The scientific substantiation of this benchmark can be found in the accompanying thesis[1].

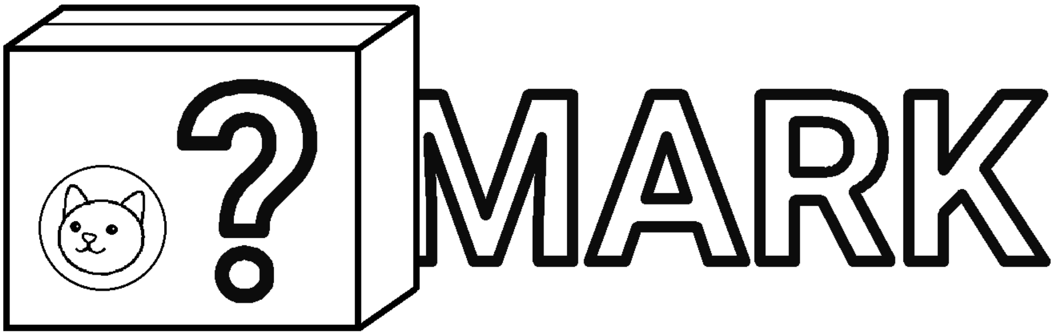


Figure A.1: The QuestionMark logo

As there is no standard available for benchmarking probabilistic databases, QuestionMark aims to cover a wide range of aspects of the tested probabilistic database management system (DBMS). This benchmark provides a convenient way to test various probabilistic database management systems and get insights on their performance. Since the queries provided in this benchmark are written in a pseudocode like language, queries can easily be translated to any probabilistic query dialect. Additionally, it provides clear guidance on how the parameters can be adapted to approach any

[1] Zandbergen, N. (2023) *QuestionMark: Designing a benchmark for probabilistic databases*. M.Sc. Thesis, University of Twente.

real-world application as close as possible. QuestionMark © 2023 by the University of Twente is licensed under Attribution 4.0 International

For the benchmark, two Python programs have been developed. Both these programs — The Dataset Generator and The Probabilistic Benchmark — need to be run to execute the benchmark. Section A.1 describes the dataset generator. Section A.2 describes the benchmark. The step by step instructions provided in this manual can also be found in `MANUAL.md` in the respective python program. This benchmark natively supports the probabilistic database management systems MayBMS and DuBio. Running the benchmark with any other probabilistic DMBS requires manual adaptation of these programs. More details on how to do this are provided in the QuestionMark Python programs and in Section A.6.

If you want to use this benchmark, allow for a total running time of three to eight hours, depending on the dataset size and included uncertainty. The benchmarking procedure consists of the following phases:

1. (approx. 60 minutes). Reading through this manual and understanding the product.
2. (30 to 180 minutes). Running QuestionMark: The Dataset Generator.
3. (30 to 90 minutes). Running QuestionMark: The Probabilistic Benchmark.
4. (approx. 60 minutes). Digesting the results and drawing conclusions.

When adding a new non-supported DBMS, the implementation of changes require an additional fifteen hours. Including a new non-supported non-PostgreSQL based DBMS, another additional fifteen hours should be taken into account.

A.1. QuestionMark: The Dataset Generator

QuestionMark: The Dataset Generator is a Python program that generates the dataset required for running QuestionMark: The Probabilistic Benchmark. This program can be downloaded from

https://github.com/utwente-db/QuestionMark/tree/thedatasetgenerator

This dataset generator prepares the dataset required for running the benchmark test. During the dataset generation phase, a dataset is produced that approaches the real-world scenario for which this benchmark is run. For this, parameters can be tuned. This program follows the general product matching workflow, which is as follows.

1. *Data Preparation.* The data is standardised and cleaned. A uniform data structure is applied.
2. *Search Space Reduction.* Since the time needed for evaluating all possible combinations grows exponentially with the dataset size, the search space for possible matches needs to be reduced to allow for efficient matching.
3. *Attribute Value Matching.* The similarity of the remaining data tuples is determined using a syntactic and semantic means, which produces a comparison vector per data attribute.

4. *Classification.* A decision model then determines the similarity score of a data tuple. This score is compared to the set thresholds to determine whether it is a matching tuple, possibly matching tuple, or non-matching tuple.
5. *Verification.* The performance of the applied product matching algorithm can be verified using standard performance metrics.

A.1.1. The Dataset Generator Roadmap

After having downloaded the program from GitLab, the generation of the probabilistic dataset can begin. When completing the listed steps, the dataset required to run QuestionMark: The Probabilistic Benchmark is obtained.

1. *Downloading the WDC datasets.* First, the base dataset should be downloaded. For additional details on this dataset, see Section A.1.2. To include this dataset in the program, create an empty folder `datasets` in the main project repository. Next, go to the WDC dataset website, scroll to the bottom of the page and download `offers_corpus_english_v2.json.gz` and `all_gs.json.gz`. Include these in the empty datasets folder. If desired, you could also download the samples from the WDC dataset website to get an impression of the dataset.
2. *Preparing the dataset generator.* To create a dataset that approaches a given real-world scenario as much as possible, first the parameters of the generator need to be set. For an explanation on these parameters, see Section A.1.3. When these parameters are set, a database connection should be established. The database of choise should be running and accepting connections. To connect QuestionMark: The Dataset Generator to your database, create a new file called `database.ini` and fill in the credentials according to the defined structure in `database.ini.tmpl`. QuestionMark: The Probabilistic Benchmark is created with PostgreSQL-based Database Management Systems in mind. In case the DBMS you want to benchmark is not PostgreSQL-based, please see Section A.6.2. If you want to benchmark a system other than DuBio or MayBMS, please see Section A.6.1.
3. *Running QuestionMark: The Dataset Generator.* Once the preparations are done, the dataset generation can begin. To run the benchmark, go to *manual.py* and run the script.
4. *What the benchmark does.* During the process of generating the dataset, several phases will be passed. If it is indicated that a smaller dataset will be used, this new dataset is produced first. To do this, a pseudo-random selection of offers is chosen from the dataset. This ensures that the same dataset will be produced each time the benchmark is run on a specific size. Next, this dataset is sorted and a dictionary is created for easy lookup. The offers present in the dataset are then put in blocks. For this, two blocking algorithms are available. First creating blocks reduces the time required to evaluate if offers should be put in the same cluster. More information on this process can be found in Section A.1.4. After the blocks are created, all offers in a block are matched and provided with a probability score. This probability indicates the likelihood that the offer belongs in a cluster, and whether its attributes are likely the correct ones. More informa-

tion on this process can be found in Section A.1.5. When the clusters are created, a database representation is created and the offers are added to a probabilistic DBMS. Finally, some preparatory queries are run.

5. *Continue with benchmarking*. The dataset is prepared! Go to QuestionMark: The Probabilistic Benchmark to continue with benchmarking. Optionally, performance tests could be run before continuing the benchmarking process.
6. *Running performance tests*. QuestionMark: The Dataset Generator also comes with a performance evaluator. This evaluator can aid in setting the parameters correctly, such that the produced dataset approaches that of the real-world as closely as possible. Several performance tests have already been run. Open `performance/performance.txt` to get insights into the behaviour of the parameters and the performance of the implemented algorithms. To run the performance tests, set the `performance` parameter to true and run the script in `manual.py` again.

A.1.2. WDC Product Data Corpus and Gold Standard

This program digests the WDC Product Data Corpus and Gold Standard for Large-Scale Product Matching, Version 2.0 to generate a probabilistic dataset from it. The WDC dataset is a large public training dataset for product matching. It is produced by extracting schema.org product descriptions from 79 thousand websites, which provides 26 million product offers. Besides the full dataset, an English language subset is offered. This subset consists of 16 million product offers. This dataset is provided with a clustering. The 16 million product offers in the English subset are categorized in 10 million clusters. Each cluster contains offers of the same product found on different websites. There are roughly 8.5 million clusters with size 1, one million clusters with size 2, and 400.000 clusters with size 3 or 4. Clusters of a size greater than 80 are filtered out of the dataset, as these are likely noise.

For this benchmark, an adaptation of the English subset is used. The dataset was adapted to include a probabilistic clustering. More information on the dataset and details on why this dataset was chosen can also be found in the accompanying thesis.

A.1.3. Dataset Generator Parameters

During the dataset generation phase, there are multiple parameters that can be tweaked. The different parameters and their effect on the resulting dataset are listed below. In order to fit the benchmark to the requirements of a specific application, it should be tailored to represent this real-world scenario as closely as possible. This can be done by tweaking the various parameters of this benchmark. The parameters ‘dataset size’ and ‘upper phi and lower phi’ have a high influence in the extent to which the produced dataset resembles the real-world application for which this benchmark test is run.

- *DBMS*. Determines into which database management system the generated dataset should be loaded and what preparatory queries need to be run.
- *Dataset size*. Determines the amount of offers included in the dataset. A percentage of the dataset can be determined to two decimal places. The offers for

the new smaller dataset are pseudo-randomly chosen, so that the same dataset is returned for multiple runs. This ensures reproducibility of the results. The full dataset contains 16 451 499 offers. The smallest dataset that can be generated is 0.01% of the full dataset, which produces an initial dataset of 1653 offers. Choose this value to generate a dataset with a size similar to that of the dataset being digested by the real-world application.

- *Whole clusters.* Determines whether the offers chosen from the larger dataset to include in the new smaller dataset are pulled from entire clusters or not. Including entire clusters increases the uncertainty of the data.

- *Word distance measure.* Determines the manner in which the distance between two words or sentences is calculated. This measure is used during the blocking phase on the attributes determined as Blocking Key Values and on all suitable attributes during the matching phase. The implemented distance measures are Levenshtein, Jaro, Jaro-Winkler, Hamming and Jaccard.

- *Blocking key values.* Determines the attributes that are included to determine the similarity of two offers during the blocking phase. Including more attributes provides a better blocking performance, but at the cost of a higher runtime.

- *Blocking similarity threshold.* Value between 0 and 1 that represents the distance between two offers. Evaluated offers with a distance lower than the threshold are included in the same block.

- *Blocking window size.* Determines the size of the sliding window. Within a window, the distance between the first and last offer is determined. This value influences the runtime.

- *Maximum block size.* Poses a restriction on the block size. Increasing this value improves the performance. As the matching phase includes a calculation with factorial time complexity, this size should not exceed six. Five is advised.

- *Matching attributes.* Determines the attributes that are used to obtain the distance between two offers during the matching phase. Including more attributes improves the performance, but increases the runtime.

- *Matching attribute weights.* Determines the weight of each attribute to calculate the final distance score. This can be tweaked to improve the performance. It has no effect on the runtime.

- *Upper phi and Lower phi.* Determines the upper and lower threshold of the distance measure. If the distance between two offers is greater than the upper phi, the two offers are certainly not the same product. If the distance is smaller than the lower phi, the two offers are certainly the same. Increasing the gap between the values ensures less false matches or non-matches, but increases the computational complexity in later phases and during querying. A smaller gap can be used to artificially reduce the uncertainty in the dataset. This value should be carefully chosen, as this influences to what extent the produced dataset imitates the data being digested by the real-world application.

A.1.4. Blocking Algoritm

To obtain a time-efficient product matching, the search space for matching pairs should be reduced. Disregarding this step results in quadratic time complexity during the product matching phase. Having 16 million products in the dataset, this step is thus essential for a time-efficient product matching. For this step, *filtering* or *blocking* can be used. QuestionMark: The Dataset Generator makes use of a blocking algoritm.

A selection of two Rule-Based Blocking techniques was implemented on the dataset to verify which algorithm performed best. These are Incrementally-Adaptive Sorted Neighborhood (ASN) and Improved Suffix Array (ISA) Blocking. Tests executed using the implemented performance evaluator indicate that ASN is the best blocking algorithm for the dataset used. The ISA algoritm is still available for use.

The ASN blocking algoritm works by sliding a window to roughly determine what offers are possible matches. For this, a sorted dataset is required. During each iteration of the algoritm, a block is created. The sliding window is placed at the first offer from the sorted list that is not yet in a block. When the start of the window is set, the enlargement phase is entered. During this phase, the window will iteratively increase in size. This is a fixed increase. After each iteration, the blocking algorithm determines the similarity score of the first and last offer in the window. If the distance between the two offers is smaller than the set threshold, the window is enlarged and a new similarity score is determined. If the distance is higher, the retrenchment phase is entered. During the retrenchment phase, the sliding window will decrease one offer in size and calculate the similarity score between the first offer in the window and the new last offer. Once the similarity score rises above the threshold, the block is created.

A.1.5. Matching Algoritm

During this phase, either an algorithm based approach or a machine learning based approach could be used. When product matching with either approach, the matching can be performed only on the product title or on all available information, i.e. including the product attributes. Only using the product title provides simplicity and speed, but at the cost of a lower precision.

For QuestionMark: The Dataset Generator, the Attribute-Based Entity Resolution approach is used as the foundation of the implementation. For this research, a comparison vector is generated from all attributes of an offer. Within each block, all possible offer combinations are generated and the distance between these offers is then provided by the vector. For simplicity, this vector is combined to a single distance score. The weight of each attribute for this final score is adaptable.

As the benchmark designed in this research is based on probabilistic data, an additional layer had to be build on top of the basic algorithm to include a probabilistic model in the final clustering. The creation of the various probabilistic clusters is based on the possible worlds model. For each block, a matching graph is created and the matching score of each edge is evaluated. Blocks containing only a single offer are always true, so are submitted to a cluster directly. When a block contains multiple offers, their matching score is evaluated. Here exists three possibilities:

- the matching score of their edges all lie above the upper threshold;

- the matching score of their edges all lie below the lower threshold;
- there are one or multiple edges between the two thresholds.

In the first case, the cluster is certain; there is only one possible world. In this case the full block becomes a new cluster. There does exist uncertainty between the correct value of the attributes, as these are likely different. In the second case, the cluster is also certain, as all offers are certainly different. In this case, each offer is put in a separate cluster. In the final case, world graphs should be constructed of the possible worlds. The amount of possible worlds created equals 2^n where n equals the amount of offers connected by an uncertain edge. When these world graphs are created, the inconsistent worlds are removed and the remaining worlds are included as different options for the same cluster. If an offer is certainly present in all worlds, this offer is added later to all generated world graphs. If an offer is certainly not present in all worlds, a separate cluster is created.

For the creation of the possible worlds, a naive implementation is used based on the theory presented in the accompanying thesis. Because of that, the space complexity for the creation of the possible worlds is factorial. This imposes a limit on the block size that can be digested by this algorithm. This imposes a maximum of six offers per block.

A.2. QuestionMark: The Probabilistic Benchmark

QuestionMark: The Probabilistic Benchmark is a Python program that runs a benchmark test for any probabilistic database management system. This program can be downloaded from

https://github.com/utwente-db/QuestionMark/tree/theprobabilisticbenchmark

This benchmark uses the dataset generated using QuestionMark: The Dataset Generator. Again, if a DBMS will be used that is not natively supported, the program needs to be adapted to allow its support. For this, see Section A.6.

A.2.1. The Probabilistic Benchmark Roadmap

After having downloaded the program from GitLab, the benchmark execution can begin. To properly run the benchmark, the following steps need to be followed. It is assumed that the process of QuestionMark: The Dataset Generator has been finished successfully and the dataset is available in a Database Management System that is accepting connections.

1. *Prepare the benchmark*. To connect to the dataset generated by QuestionMark: The Dataset Generator, create a file called `database.ini` and fill enter the credentials according to the defined structure in `database.ini.tmpl`. Then, the parameters for running the benchmark must be set. For an explanation on these parameters, see Section A.2.2. Again, if a new DBMS or a non-PostgreSQL based DBMS will be benchmarked, please follow the steps listed in Section A.6. To test the connection to the database, set the parameter `test` to True. Remember to set this value to False before running the benchmark test.

2. *Run the benchmark.* The benchmark is now fully prepared to be run. To run the benchmark and obtain the results, go to `manual.py` and run the script. For additional details on the queries included in the benchmark, see Section A.3.
3. *Reading the results.* When the benchmark execution is finished, the results can be viewed. The benchmark results are stored in `QM_metric_ results.txt` and `QM_query_results.txt`. Both files provide insights into the performance of the tested benchmark. For more instructions on how to digest and interpret the results, see Section A.4.

A.2.2. Benchmark Parameters

During the benchmarking process, there are also parameters that can be tweaked. The parameters during this phase are mostly related to the DBMS used and the functionality coverage of the benchmark.

- *DBMS.* Determines the database management system that will be used for the execution of the benchmark. Additional systems can be added when support for them is also added to the benchmark program.
- *Iterations.* Denotes the amount of times a query is run to obtain a runtime average from the queries. This is a global variable that is used for all queries. Increasing this number will provide a more precise outcome of the average run time, but at the cost of a longer benchmark execution time. The total amount of iterations is always +1 to create a warm start.
- *Show Query Plan.* Boolean value. If true, the query plan for each query is also provided with the benchmark result. Enabling this variable does not influence the execution time of the queries.
- *Timeout.* Ensures that queries that take too long to return an answer will be aborted. Once a query times out, this will be noted in the benchmark result and the next query is started. The current implementation abruptly stops the benchmark execution.
- *Queries.* A list that contains all queries from the benchmark. Depending on the goal of the benchmark run, queries that are not relevant can be removed from the benchmark run. Removing queries lowers the total time required to run the benchmark and focuses the results to what is important. The benchmark can also be run in several iterations, as to create several smaller, more focused benchmark results.

A.3. Benchmark Queries

This section discusses the queries included in the benchmark. The list of queries included in the QuestionMark benchmark can be found in Appendix A.7.

A.3.1. Queries

QuestionMark: The Probabilistic Benchmark offers a range of queries that can be used to test various types of systems. The queries are selected to cover the diverse possibilities of the dataset, but also include functionalities that are key to some more well-known probabilistic database management systems. The queries are sub-divided into queries that provide insight into the dataset, probabilistic queries that could be run more frequently and insert-update-delete queries. The table below provides a quick overview of the included queries.

test 1	Simple query to test the connection.
insight 1	Retrieves the full dataset, gain insight in data structure and load handling.
insight 2	Provides insight into the dataset and probability handling.
insight 3	Provides insight into the distribution of cluster volumes.
insight 4	Gets the percentage of certain clusters.
insight 5	Gets the id and probability of the offers with a specific variable value or sentence.
insight 6	Gets the average probability of the dataset.
probabilistic 1	Gets offers with the probability of their occurrence.
probabilistic 2	Gets the expected count of the categories.
probabilistic 3	Gets the expected sum of the product ids per cluster.
probabilistic 4	Gets the sentence and probability per category.
probabilistic 5	Returns the most probable offer that is related to a specified string.
probabilistic 6	Returns all offers containing a specified string with a high uncertainty so these can be classified by human inspection.
insert update delete 1	Inserting a single row.
insert update delete 2	Inserting bulk.
insert update delete 3	Updates uncertainty.
insert update delete 4	Removes uncertainty.
insert update delete 5	Deletes a cluster.

A.3.2. Altering Queries

The queries presented in this benchmark are already translated and included in QuestionMark: The Probabilistic Benchmark in the dialects of DuBio and MayBMS. Please note that these query implementations are written with a dataset of 0.01% size. When producing a dataset of a different size, it could happen that the clusters used in those queries are not present in the produced dataset. It is thus of importance to always check the queries before running the benchmark. The following queries require special attention:

- *Query insight 5.* This query requires a specific variable or sentence to be defined. You could either define one that does not exist in the database, or choose one that does exist.
- *Queries probabilistic 5 and probabilistic 6.* This query uses pattern matching to obtain a selection of offers that satisfy that pattern. It is advised to query for anything that exists in the dataset.
- *Query insert, update, delete 3.* This query requires a specific cluster to be defined. Seek for any cluster of size four. Include its ID in the query and change the probability with variables accordingly.

- *Query insert, update, delete 4.* This query should also be run on any cluster of size four. Include the ID of each offer present in that cluster in one of the four queries. Include the cluster ID in the probability variables.
- *Query insert, update, delete 5.* This query removes a cluster. Search for a cluster with a sufficiently large size and include its ID. With the current limitations, a cluster with the largest size is advised.
- *Queries timing out.* During the benchmarking, it could happen that queries take too long to return an answer. In that case, the query is timed out. To verify whether the functionality of the query is supported, change the query to run on the 'part' table. This part table contains a small portion of the dataset. If the query still times out with this table, it could be worthwhile to decrease the size of this table even further. To do this, go to QuestionMark: The Dataset Generator and open `database_filler_[DBMS].py`. Then reduce the value in `LIMIT FLOOR()` in the first query of `prep_queries`.
- *Queries raising exceptions.* During the benchmarking, it could also happen that queries throw errors. When any query raises the exception `invalid memory alloc request size 1073741824` or `Ran out of memory retrieving query results`, it can also be worthwhile to run the query on the 'part' table. Most likely, reducing the dataset size that the query needs to digest removes this specific error. This verifies whether the functionalities in the query are supported by the system or not. It is worthwhile so remain critical when errors are thrown, sometimes a workaround can be found to still find a fix. Another option is to run the query on a database tool, as it could be that the DBMS cannot handle some requests from `psycopg2`.

A.4. Produced Results

After running QuestionMark: The Probabilistic Benchmark it is time to analyse the produced results. The benchmark provides information about the benchmark through the following metrics:

- The brevity of the query dialect.
- The query functionality coverage.
- The runtime of the queries.
- The probabilistic data overhead.
- The user friendliness of the system.

The benchmark produces two files when run: `QM_metrics_results.txt` and `QM_query_results.txt`. These two files contain the raw metric values that can be digested to obtain valuable information from. The metrics provide information on the effectiveness, efficiency and appeal of the tested software. `QM_metrics_results.txt` provides the raw data of four metrics in the main results part and provides an overview of all errors thrown while running the benchmark. If no errors were thrown, nothing will be printed and only the main part is visible. In `QM_query_results.txt`, all queries, their results, and their execution time are shown.

A.4.1. Metrics

The benchmark thus produces data for five metrics. This section provides additional details on each of these metrics.

Brevity of the query dialect. This metric is determined by the total amount of characters needed for all queries and gives insights into the succinctness of the query language. A more succinct query dialect often requires less time to write queries with and is often easier to understand. This metric value is obtained by iterating over all queries and adding their character count. Spaces are removed from the calculation. Optionally, characters can be removed from specific queries. For example in query `IUD_1_rollback` offers are added to the database. As the data that represents the offer is not indicative of the complexity of the query language, the amount of characters used for that representation is subtracted from the total character count for that query.

Query functionality coverage. This metric provides insight into the functionality coverage of the database system and is determined by multiple sub-metrics. When running the queries to obtain their results and runtime, it can happen that a specific functionality is not supported or the database system cannot handle the load required to execute the query. In these cases, the system returns an error. The error raised during execution are stored and printed as the query result. After the benchmark execution has finished, an overview table is created that indicates what queries finished execution and which threw an error. The percentage of successful queries is then also determined. For each query that threw an error, it also indicates what query functionality might be lacking. In each case, a critical look is needed to verify whether the error is thrown due to an actual lack of functionality support or due to another reason, for example a typo. With the gathered knowledge, the functionality coverage table can be manually filled in. In this table, a distinction is made between functionality that is natively supported and functionality that can be implemented with a workaround method

Runtime of queries. This metric provides insight into the speed of query execution. A lower runtime is required to obtain higher query throughput rates and improves the flow of business processes relying on the query results. This metric is also obtained by a combination of sub-metrics. To obtain the runtime of a query, the PostgreSQL `EXPLAIN ANALYSE` statement is used. This statement returns the execution plan of various queries or statements and tracks its runtime. When available, it differentiates between the planning time and execution time of a query. In this distinction is not supported by the DBMS, only a total runtime is returned. For each query, the average runtime over the specified iterations is printed. Each query is run with a warm start. After all benchmark queries have run, a total average planning time and execution time, or total average runtime is calculated. This is the sum of all time averages of all queries. The total time provides a quick idea of the speed of the tested DBMS. For each application scenario, the acceptable runtime of a query differs. It is thus advised to verify the significance of the queries and per query determine the acceptable runtime.

Probabilistic data overhead. This metric represents the additional storage space required to store the probabilistic representation of the data. When processing large volumes of data, needing additional storage space to store the probabilistic representation of the data could get costly. As each probabilistic DBMS stores their probabilistic representation in a unique way, the probabilistic data overload is calculated for each DBMS differently. For both systems, the storage space used is determined by the `pg_size_pretty` statement of PostgreSQL. For DuBio, the overhead percentage is determined using the following calculation:

$$\frac{sentence + dictionary}{offers + dictionary} \times 100$$

Here, *sentence* is the size `_sentence` column in the `offers` table, *dictionary* is the size of the `_dict` table, and *offers* is the size of the `offers` table.

For MayBMS, the following calculation is used to determine the overhead percentage:

$$\frac{setup \times (1 - \frac{distinct\ ids\ count}{ids\ count}) + (offers - setup)}{offers} \times 100$$

Here, *setup* is the size of the `offers_setup` table, *distinct_ids_count* is the count of all distinct values of the `id` column in the `offers` table, *ids_count* is the count of all values of the `id` column in the `offers` table, and *offers* is the size of the `offers` table.

The calculation for MayBMS is a bit more complex, as MayBMS does not create a compact representation of the probability space over a single offer. Because of that, data duplication is created in the offers table. The overhead that this duplication creates is determined by counting the `id` values.

User friendliness. User friendliness is another metric that is composed from several sub-metrics. As user friendliness is something of a more personal taste and cannot be measured from a benchmark run, all sub-metrics are in the form of statements that should be rated on a scale from 1 to 5, 1 meaning that the statement is not true, an 5 meaning that it is very much true. The following aspects should be evaluated to determine a final user friendliness score of the system:

[1, 2, 3, 4, 5]	The software is well documented.
[1, 2, 3, 4, 5]	The software was easy to work with.
[1, 2, 3, 4, 5]	We have sufficient in-house expertise to work well with the software.
[1, 2, 3, 4, 5]	I am satisfied with the monetary expenses that need to be made for running the software.
[1, 2, 3, 4, 5]	The software has a support service.

A.5. Digesting the Results

To digest the raw metrics provided by the benchmark and obtain useful information from them, the benchmark performance is categorised in terms of its effectiveness, efficiency and appeal. please follow the instructions below to digest the raw results and gain insights into the performance of the system.

A.5.1. Effectiveness

The effectiveness of the software relates to the quality of fulfilling the purpose. To obtain a complete picture about the effectiveness of the tested software, both generated files should be considered. To obtain a global picture on the effectiveness of the software, open `QM_metrics_results.txt`. Here, the metric ‘percentage of successful queries‘ is of importance. Ideally, this value will be 100%. If this is not the case, errors have been thrown during the benchmarking process. These errors are displayed at the bottom of the file. For each error, verify if it is thrown due to a lack of functionality support, or due to other reasons, such as programming or memory errors. If the error is due to any of the other reasons, try to eliminate these and run the query again. Keep track of the following information when altering queries that have thrown an error:

Fix log #	
Query that raised an exception:	
Prior adaptations done on the query:	
Exception raised by the query:	
Suspected cause of the exception:	
Implemented fix:	

If any error cannot be fixed, the functionality that is required is lacking in the tested software. In the errors overview, a list of possible functionality gaps is listed below each error. Verify if the error is caused by any of these and note the missing functionality. With this, the coverage of query functionality can be identified. Below is a list of the functionalities that are identified. This list can be expanded when different functionalities are of importance.

As a final step, open `QM_query_results.txt` and verify the results returned by the queries. This step is optional, as there is no truth table provided with the benchmark. If anything strange is shown, verify if the tested software is performing badly.

#	Native	Possible	**Functionality**
1	[ ]	[ ]	Support of most recent deterministic DBMS queries
2	[ ]	[ ]	Offering a compact representation of the present uncertainty
3	[ ]	[ ]	Get the probability of an offer
4	[ ]	[ ]	Get the probability of a composed result
5	[ ]	[ ]	Apply aggregate functions on probabilities
6	[ ]	[ ]	Filtering on probability
7	[ ]	[ ]	Get the expected count
8	[ ]	[ ]	Get the expected sum
9	[ ]	[ ]	Get the most probable answer
10	[ ]	[ ]	Verify if a specific possible world exists
11	[ ]	[ ]	Verify if a record is certain
12	[ ]	[ ]	Updating the uncertainty of an offer
13	[ ]	[ ]	Repair the probability space after addition, update or deletion of offers
14			Any anomalies discovered during benchmarking

The following table can be used as a guide to see what queries require what functionality:

#	**Functionality**	**Queries**
1	Support of most recent deterministic DBMS queries	Any
2	Offering a compact representation of the present uncertainty	Insight 2
3	Get the probability of an offer	Probabilistic 1
4	Get the probability of a composed result	Insight 5; Insight 6; Probabilistic 4
5	Apply aggregate functions on probabilities	Insight 4; Probabilistic 4
6	Filtering on probability	Probabilistic 6
7	Get the expected count	Probabilistic 2
8	Get the expected sum	Probabilistic 3
9	Get the most probable answer	Probabilistic 5
10	Verify if a specific possible world exists	Insight 5
11	Verify if a record is certain	Insight 4
12	Updating the uncertainty of an offer	Insert, Update, Delete 3
13	Repair the probability space after addition, update or deletion of offers	Insert, Update, Delete 1; Insert, Update, Delete 4
14	Any anomalies discovered during benchmarking	Any

A.5.2. Efficiency

The efficiency of the software relates to its use of resources and execution speed. To obtain a complete picture of the efficiency of the testes software, several metrics should be evaluated. The most prominent efficiency metric is the speed of the tested software. The speed of the software can be collected by both the total average execution time and the time per query. The desired speed is fully dependent on your requirements. If specific types of operations are most important for the real-world software, open `QM_query_results.txt` and verify if the queries containing that functionality have an acceptable execution time. A visual representation of the execution times of the queries can be found in `results/graphs/QM_graph_runtime.png`. To ensure a clear presentation of the visual results, it might be beneficial to run the benchmark over several subsets of queries. One could, for example, make a separate fun for all insert, update and delete statements, or remove all queries with a significantly higher execution time.

Another indication of efficiency is the amount of characters needed for the queries. It is your decision if this metric is of importance. This metric is also related to the appeal of the software. Query dialects that require more characters are possibly more difficult to understand and possibly require more time to define queries with. For this sub-metric, also a visual representation is included. This can be found in `results/graphs/QM_graph_runtime.png`. Again, if the produced graph is hard to read it might be beneficial to run the benchmark over several query subsets.

A final indication of efficiency is the additional storage space required to store the probabilistic representation of the data. If storage is sparse, having a system that requires less storage for the probabilistic representation is better. Please verify what overhead is acceptable. This sub-metric is only indicated as a single percentage in `QM_metrics_results.txt`

A.5.3. Appeal

The appeal of the software relates to the human element, including the satisfaction of use. Whether the tested software appeals to the company is thus more about personal preference. To guide with answering the question if the tested software is appealing, a list of statements is defined. The score given to these statements define the appeal score of the software. Rate each of the statements below with a score from 1 to 5. A 1 means that the statement does not align with your personal opinion on the software, so that you strongly disagree with the statement, whereas 5 means that the statement is very much true, so you strongly agree. Scoring a 3 provides a neutral opinion.

[1, 2, 3, 4, 5]	The software is well documented.
[1, 2, 3, 4, 5]	The software was easy to work with.
[1, 2, 3, 4, 5]	We have sufficient in-house expertise to work well with the software.
[1, 2, 3, 4, 5]	I am satisfied with the monetary expenses that need to be made for running the software.
[1, 2, 3, 4, 5]	The software has a support service.

A.5.4. Drawing Conclusions

After information on each metric is collected, conclusions can be drawn from this newly acquired information. As a first step, verify if the software supports all functionalities required. A software that cannot run your key processes is practically useless. If this is satisfied, the importance of each metric should be identified. When all metrics are ordered by their importance, a better picture of the suitability of the software can be drawn. Be critical of your requirements and if the tested software fits these well enough. If two systems are benchmarked, compare their results.

A.6. Including Other Database Management Systems

The QuestionMark benchmark for probabilistic databases was designed with generalisability in mind. Hence why all benchmark queries are also provided in a pseudocode like language. To test the system, two promising probabilistic database management systems are already supported in the system; MayBMS and DuBio. Both these systems are based on PostgreSQL, hence why the QuestionMark Python programs are also written with PostgreSQL based systems in mind. If you want to benchmark another probabilistic database system, both Python programs need to be adapted to fit the new system. For this, additional changes are required when implementing a new non-PostgreSQL based system.

A.6.1. Including any new Probabilistic DBMS

When including a new PostgreSQL based DMBS, no alterations need to be made to the existing codebase. However, new functions should be defined based on the existing codebase.

As each probabilistic DBMS has its own unique structure when it comes to representing the probabilities and/or sentences of the possible worlds, the dataset generation should be adapted to fit the requirements of the new systems. For this, new functions should be defined. For ease, the placeholder `NAME` will be used, which denotes the name of the newly added DBMS. The following additions should be made to QuestionMark: The Dataset Generator.

1. `database_filler_NAME.py`. As the dataset needs to be properly prepared for the new DBMS, functions need to be designed to tailor the produced dataset to the needs of the DBMS. The structure should be similar to that defined in `database_filler_dubio.py` and `database_filler_maybms.py`.

A new probabilistic DBMS also has its own SQL dialect, so in QuestionMark: The Probabilistic Benchmark additions should also be made to the code.

1. `queries_NAME.py`. To include a new DBMS, the first step is to include the queries in the corresponding dialect. To do this, create `queries_NAME.py`. To see what queries should be included, `queries_pseudo_code`, `queries_MayBMS.py` and `queries_DuBio.py` can be used as a translation guide. Please stick to the structure used in these files. When the proficiency level of the to be included query dialect is not sufficiently high, it is advised to first test the queries in a database tool of preference. This makes debugging queries easier.
2. `execute_queries.py`. This file is responsible for sending the queries to the DBMS. In this file, include `from queries_NAME import NAME_QUERIES_DICT` . In `execute_query()`, also include the DBMS in the first if-statement. Finally, check if the execution time returned by the DBMS follows the pattern from MayBMS or from DuBio. When the DBMS uses PostgreSQL 10 or higher, the default can be used.
3. `output_tui.py`. This file prints the benchmark output. In `create_result_`

file(), add the new DBMS in the if-statement.

4. parameters.py. Include the new DBMS as an option of the DBMS variable.
5. metrics.py. To obtain the metric values from the new system, some metrics should be tailored to the system. In char_count(), add the DBMS in the if statement and verify if specific queries should get a discount in character count. This is done, since characters required for raw record information do not count towards the complexity of the dialect. Also the calculation in prob_size() should be adapted and tailored to the manner in which the DBMS stores the probabilistic data.

A.6.2. Including a new Non-PostgreSQL Based DBMS

To include a new non-PostgreSQL based DBMS system, additional steps need to be taken. The following adaptations should be made to QuestionMark: The Dataset Generator.

1. database.ini.tmpl. Needs to be adjusted to support a different system.
2. database_filler.py. Most of this file should be adapted to generate a connection to the DBMS. Right now, the program uses psycopg2 to establish a database connection. This library only works with PostgreSQL based database systems. To provide support for other systems, please read through all methods in this file and make adaptations where required.
3. insert_query.py. The create() method also uses Psycopg2. This method should thus also be changed.

The following additional adaptations should be made to QuestionMark: The Probabilistic Benchmark.

1. execute_query.py. This file establishes the connection with the database and runs the benchmark. Please read through all functions and change the code where needed.
2. connect_db.py. This file also establishes a connection with the database and is used for metric queries. Please read through all functions and change the code where needed. Most of these alterations can be copied from those implemented in execute_query.
3. database.ini.tmpl. Needs to be adjusted to support a different system.
4. run_benchmark.py. This is the main file to run the benchmark. Also here, psycopg2 is used. The present function thus needs to be adapted.

A.7. Query Implementations

This Appendix contains the benchmark queries in pseudocode SQL and provides examples of its implementations into the dialects of DuBio and MayBMS.

A.7.1. Queries in Pseudocode

The queries below are included in the benchmark. For each query, additional information is provided on its functioning and why it is included in the benchmark.

Test 1: Testing the connection. The first query is mainly included to have a low strain query that can be used to test the connection. This query consists of basic SQL-functionalities and all systems should be able to run this.

```
select attribute 'id'
from entity 'offers'
return the first 10 records;
```

Insight 1: Retrieve the full dataset, gain insight in data structure. This query is a real strain tester of the system. The query itself is simple, but it requires the DBMS to return all its data.

```
select all attributes
from 'offers';

if present select all remaining data;
```

Insight 2: Provide insight into the concentration of offers. This query can be used to verify to what extent the DBMS can concentrate the uncertainty of an offer. It also provides insights into the number of clusters that have been formed.

```
select the count of all attributes alias 'records',
    ↪ the count of all distinct values of attribute 'id' alias 'offers',
    ↪ the count of all distinct values of attribute 'cluster_id' alias '
    ↪ clusters'
from entity 'offers';
```

Insight 3: Provide insight into the distribution of cluster volumes. This query is included as lower strain deterministic query and also includes useful insight into the dataset. As larger clusters put more strain on probability calculations, it is useful to gain insight into the distribution of cluster volumes.

```
select attribute 'cluster_size',
    ↪ the count of all values of attribute 'cluster_size' alias 'amount'
from subquery (
    select the count of all distinct values of attribute 'id' alias '
        ↪ cluster_size'
    from entity 'offers'
    grouped by attribute 'cluster_id'
) alias 'cluster_sizes'
grouped by attribute 'cluster_size'
ascendingly ordered by 'cluster_size';
```

Insight 4: Gets the percentage of certain clusters. This query provides insight into the uncertainty of the generated dataset. A new dataset can be generated when the result of this query does not match the uncertainty of the real-world dataset. It also verifies if probability calculations can be done on aggregated data.

```
select the count of all certain records divided by the count of all
    ↪ attributes times 100 rounded to four decimal places alias 'certain
    ↪ percentage'
from entity offers;
```

Insight 5: Get the id and probability of the offers from a specific possible world. In some situations it might turn out useful to make a selection based on the probability space of a record. This query returns any record satisfying a specific sentence or probability space declaration.

```
select attribute 'id',
   ↪ the probability attribute,
   ↪ the variable or sentence attribute
from entity 'offers'
satisfying a specific variable or sentence statement;
```

Insight 6: Get the average probability of the dataset. This is another query that tests the strain of the system. It performs a probability calculation over the entire dataset. It also provides insights into the uncertainty of the dataset.

```
select the average of the probability attribute rounded to four decimal
   ↪ places alias 'certainty_of_the_dataset'
from entity 'offers';
```

Probabilistic 1: Get offers with the probability of their occurrence. This query contains the most basic added functionality of any probabilistic DBMS, which is the presentation of the probability. It also evaluates the speed of ordering based on the probability attribute.

```
select the probability attribute rounded to four decimal places alias '
   ↪ probability',
   ↪ all attributes
from entity 'offers'
descendingly ordered by 'probability';
```

Probabilistic 2: Gets the expected count of the categories. One more advanced operation on probabilistic data is to obtain the expected count of an attribute. This query evaluates if that operation is supported.

```
select the attribute 'category',
   ↪ the expected count per attribute 'category' alias 'expected_count'
from entity 'offers'
grouped by attribute 'category'
descendingly ordered by 'expected_count';
```

Probabilistic 3: Gets the expected sum of the product ids per cluster. Another closely related operation is the expected sum. This query evaluates if that operation is supported.

```
select attribute 'cluster_id',
   ↪ the expected sum per attribute 'id' alias 'number_of_offers'
from entity 'offers'
grouped by attribute 'cluster_id'
descendingly ordered by 'number_of_offers';
```

Probablistic 4: Gets the variables/sentence and probability for the categories. This query is again focused on strain testing. This query produces large aggregations of probabilities, which need to be evaluated to return the query result. This query tests if the DBMS can digest these large aggregations.

```
select attribute 'category',
   ↪ the compound variable/sentence attribute,
   ↪ the compound probability attribute rounded to four demical places
   ↪ alias 'probability'
from entity 'offers'
grouped by attribute 'category'
descendingly ordered by 'probability';
```

Probabilistic 5: Returns the most probable offer that is related to a specified string . This query represents the behaviour of a search engine, where the most probable offer satisfying a search condition should be returned. An example string is ’card’. The pseudocode provided contains a workaround method to obtain the most probable answer. It can be shortened to represent native support of this functionality. This query contains hard-coded information and may require an adaptation when having generated a fitting dataset. See Section 5.4.1 for more information.

```
select all attributes,
   ↪ the probability attribute rounded to four decimal places alias '
   ↪ probability'
from entity 'offers'
satisfying that the attribute value 'cluster_id' exists in subquery (
        select attribute 'cluster_id'
        from entity 'offers'
        satisfying that attribute 'title' a specified string
                or that attribute 'description' contains a specified
                   ↪ string
)
descendingly ordered by 'probability'
return the first 1 records;
```

Probabilistic 6: Returns all offers containing a specified string with a high uncertainty. When a dataset contains large volumes of highly uncertain data, it can be useful to let a selection of data pass human inspection. This query returns the most uncertain offers so these can be manually classified. This query contains hard-coded information and may require an adaptation when having generated a fitting dataset. See Section 5.4.1 for more information.

```
select attribute 'id',
   ↪ attribute 'cluster_id',
   ↪ attribute 'brand',
   ↪ attribute 'category',
   ↪ attribute 'identifiers'
from entity 'offers'
satisfying that attribute 'title' contains a specified string
        or that attribute 'description' contains a specified string
        and the value of the probability attribute is higher than 0.45
        and the value of the probability attribute is lower than 0.55;
```

Insert, Update, Delete 1: Inserting a new probabilistic cluster. When dealing with probabilistic databases, new data can be added regularly. This query verifies the speed at which new clusters can be added to the database.

```
insert into entity 'offers'
the values (a copy of a cluster with size five, with negative id values.);
```

```

if required add the new probabilities to the corresponding entity;
if required manually repair the probability space;
```

Insert, Update, Delete 2: Inserting bulk. When large volumes of data are constantly added to the database, they are likely added in bulk. This query strain tests the DBMS on large additions of probabilistic data. The current table 'bulk insert' contains 1000 offers and their corresponding probabilities.

```
insert into entity 'offers'
the results of subquery (
        select all attributes
        from entity 'bulk_insert'
);

if required add the new probabilities to the corresponding entity;
if required manually repair the probability space;
```

Insert, Update, Delete 3: Update uncertainty. This query updates the uncertainty of a specific cluster. As the location of the probability greatly determines the form of this query, its pseudocode is more abstract. This query contains hard-coded information and may require an adaptation when having generated a fitting dataset. See Section 5.4.1 for more information.

```
update the entity containing the probabilities.
alter half of the probabilities of a cluster with four offers;

if required manually repair the probability space;
```

Insert, Update, Delete 4: Remove uncertainty. When working with probabilistic data, chances are that new evidence will be found and the database should be updated accordingly. In this query, a cluster of size 4 will be split into three clusters. It is currently run on the cluster with cluster_id 162. This query contains hard-coded information and may require an adaptation when having generated a fitting dataset. See Section 5.4.1 for more information.

```
update entity 'offers'
set attribute 'cluster_id' with the maximum value of attribute 'cluster_id' + 1,
    the variable/sentence/probability attribute to certain
satisfying that attribute 'id' has the value of the first offer in the cluster;

update entity 'offers'
set attribute 'cluster_id' with the maximum value of attribute 'cluster_id' + 1,
    the variable/sentence/probability attribute to certain
satisfying that attribute 'id' has the value of the third offer in the cluster;

update entity 'offers'
set the variable/sentence/probability attribute to a new normalized value
satisfying that attribute 'id' has the value of the second offer in the cluster;

update entity 'offers'
set the variable/sentence/probability attribute to a new normalized value
satisfying that attribute 'id' has the value of the fourth offer in the cluster;

if required update the probabilities in the corresponding entity;
if required update the probability space;
```

Insert, Update, Delete 5: Delete a full cluster. Any probabilistic data should also not slow down the deletion of data significantly. This query tests the speed of the DBMS when deleting probabilistic data. This query contains hard-coded information and may require an adaptation when having generated a fitting dataset. See Section 5.4.1 for more information.

```
delete all records from entity 'offers'
satisfying that attribute 'cluster_id' has the value of the specified cluster;

if required delete the probabilities in the corresponding entity;
if required manually repair the probability space;
```

A.7.2. Queries in DuBio

```sql
-- Test 1:
SELECT id
FROM offers
LIMIT 10;

-- Insight 1:
SELECT *
FROM offers;

SELECT print(dict) FROM _dict WHERE name='mydict';

-- Insight 2:
SELECT COUNT(*) as records,
        COUNT(DISTINCT(id)) as offers,
        COUNT(DISTINCT(cluster_id)) as clusters
FROM offers;

-- Insight 3:
SELECT cluster_size, COUNT(cluster_size) as amount
FROM (
    SELECT COUNT(DISTINCT(id)) as cluster_size
    FROM offers
    GROUP BY cluster_id
) as cluster_sizes
GROUP BY cluster_size
ORDER BY cluster_size ASC;

-- Insight 4:
SELECT ROUND(COUNT(CASE WHEN istrue(_sentence) THEN 1 END)::decimal /
   ↪ COUNT(*)::decimal, 4) * 100 AS certain_percentage
FROM offers;

-- Insight 5:
WITH prob AS (
        SELECT prob(dict, 'w43=1') AS probability
        FROM _dict
        WHERE name = 'mydict'
)
SELECT offers.id, prob.probability, hasrva(_sentence, 'w43=1')
FROM offers, prob
WHERE hasrva(_sentence, 'w43=1');

-- Insight 6:
SELECT AVG(probability) AS certainty_of_the_dataset
FROM (
        SELECT round(prob(d.dict, o._sentence)::NUMERIC, 4) AS probability
        FROM offers o, _dict d
        WHERE d.name = 'mydict'
) AS probabilities;

-- Probabilistic 1:
SELECT round(prob(d.dict, p._sentence)::NUMERIC, 4) AS probability, o.*
FROM offers o, _dict d
WHERE d.name = 'mydict'
ORDER BY probability DESC;
```

```

-- Probabilistic 2:
SELECT category, SUM(prob(d.dict, o._sentence)) AS expected_count
FROM offers o, _dict d
WHERE d.name = 'mydict'
GROUP BY category
ORDER BY expected_count DESC;

-- Probabilistic 3:
SELECT cluster_id, ROUND(SUM(id * prob(d.dict, o._sentence))::NUMERIC, 2)
    ↪ AS expected_sum, COUNT(id) AS number_of_offers
FROM offers o, _dict d
WHERE d.name = 'mydict'
GROUP BY cluster_id
ORDER BY number_of_offers DESC;

-- Probablistic 4:
WITH category_sentence AS (
    SELECT category, AGG_OR(_sentence) AS sentence
    FROM part
    GROUP BY category
)
SELECT cs.*, round(prob(d.dict, cs.sentence)::NUMERIC, 4) AS probability
FROM category_sentence cs, _dict d
WHERE d.name = 'mydict'
ORDER BY probability ASC;

-- Probabilistic 5:
Returns the most probable offer that is related to 'ford'.
SELECT p.*, round(prob(d.dict, _sentence)::NUMERIC, 4) AS probability
FROM part p, _dict d
WHERE cluster_id IN (
        SELECT cluster_id
        FROM part
        WHERE title LIKE '%ford%'
        OR description LIKE '%ford%'
)
ORDER BY probability DESC
LIMIT 1;

-- Probabilistic 6:
SELECT o.*
FROM offers o, _dict d
WHERE title LIKE '%card%'
OR description LIKE '%card%'
AND prob(d.dict, _sentence) > 0.45
AND prob(d.dict, _sentence) < 0.55;

-- Insert, Update, Delete 1:
INSERT INTO offers (id, cluster_id, title, brand, category, description,
    ↪ price, identifiers, keyvaluepairs, spectablecontent, "_sentence")
    VALUES(-464, 77, ..., Bdd('b77x1=1&v77=1') ),
          (-466, 77, ..., Bdd('b78x1=0&v78=1') ),
          (-468, 77, ..., Bdd('b77x1=2&v77=1') ),
          (-469, 77, ..., Bdd('b78x1=1&v78=1') ),
          (-471, 77, ..., Bdd('b77x1=0&v77=1') );
```

```sql

UPDATE _dict
SET dict = add(dict, 'b77x1=0:0.24454, ..., v77=3:0.246')
WHERE name='mydict';

-- Insert, Update, Delete 2:
INSERT INTO offers(id, cluster_id, title, brand, category, description, price, identifiers, keyvaluepairs, spectablecontent, _sentence)
    SELECT * FROM bulk_insert;

UPDATE _dict
SET dict = add(dict, 'b000x1=0:0.500000, ... v966=2:0.203147')
WHERE name='mydict';

-- Insert, Update, Delete 3:
UPDATE _dict
SET dict = upd(dict, 'a7x1=0:0.3992, ..., w8=3:0.184')
WHERE name='mydict';

-- Insert, Update, Delete 4:
WITH max_cluster AS (
    SELECT (max(cluster_id) + 1) AS max_id
    FROM offers
)
UPDATE offers
SET cluster_id = max_cluster.max_id,
    _sentence = Bdd('1')
FROM max_cluster
WHERE id = 2689021;

WITH max_cluster AS (
    SELECT max(cluster_id) + 1 AS max_id
    FROM offers
)
UPDATE offers
SET cluster_id = max_cluster.max_id,
    _sentence = Bdd('1')
FROM max_cluster
WHERE id = 7257664;

UPDATE offers
SET _sentence = Bdd('a162x5=0&w162=0')
WHERE id = 10198975;

UPDATE offers
SET _sentence = Bdd('a162x5=1&w162=0')
WHERE id = 2668263;

UPDATE _dict
SET dict = add(dict, 'w162=0:0.83')
WHERE name='mydict';

UPDATE _dict
SET dict = del(dict, 'a162x5=2')
WHERE name='mydict';

```

```
-- Insert, Update, Delete 5:
DELETE FROM offers
WHERE cluster_id = 41;

UPDATE _dict
SET dict = del(dict, 'a41x1=0, ..., w44=5')
WHERE name='mydict';
```

A.7.3. Queries in MayBMS

```

-- Test 1:
SELECT id
FROM offers
LIMIT 10;

-- Insight 1:
SELECT *
FROM offers;

-- Insight 2:
SELECT COUNT(*) as records,
        COUNT(DISTINCT(id)) as offers,
        COUNT(DISTINCT(cluster_id)) as clusters
FROM offers_setup;

-- Insight 3:
SELECT cluster_size, COUNT(cluster_size) as amount
FROM (
    SELECT COUNT(DISTINCT(id)) as cluster_size
    FROM offers_setup
    GROUP BY cluster_id
) as cluster_sizes
GROUP BY cluster_size
ORDER BY cluster_size ASC;

-- Insight 4:
SELECT ROUND(all_certain::decimal / all_offers::decimal, 4) * 100 AS
   ↪ certain_percentage
FROM (
        SELECT COUNT(id) AS all_offers
        FROM offers_setup
) AS count_all, (
        SELECT COUNT(id) AS all_certain
        FROM (
                SELECT id, tconf() AS confidence
                FROM offers
        ) AS confidences
        WHERE confidence = 1
) AS count_cert;

-- Insight 5:
SELECT id, tconf(*), _v0
FROM offers
WHERE _v1 = 52379
AND _d1 = 548185;
```

```sql

-- Insight 6:
SELECT round((AVG(tconf()) * 100)::NUMERIC, 4) AS certainty_of_the_dataset
FROM offers;

-- Probabilistic 1:
SELECT round(tconf()::decimal, 4) AS probability, *
FROM offers
ORDER BY probability DESC;

-- Probabilistic 2:
SELECT category, ECOUNT() AS expected_count
FROM offers
GROUP BY category
ORDER BY expected_count DESC;

-- Probabilistic 3:
SELECT cluster_id, esum(id), COUNT(id) AS number_of_offers
FROM offers
GROUP BY cluster_id
ORDER BY number_of_offers DESC;

-- Probablistic 4:
SELECT category, conf() AS probability
FROM offers
GROUP BY category
ORDER BY probability DESC;

-- Probabilistic 5:
SELECT *, round(tconf()::NUMERIC, 4) AS probability
FROM offers
WHERE cluster_id IN (
    SELECT cluster_id
    FROM offers_setup
    WHERE title LIKE '%card%'
    OR description LIKE '%card%'
)
ORDER BY probability DESC
LIMIT 1;

-- Probabilistic 6:
SELECT id, cluster_id, brand, category, identifiers
FROM offers
WHERE title LIKE '%card%'
OR description LIKE '%card%'
AND tconf() > 0.45
AND tconf() < 0.55;

-- Insert, Update, Delete 1:
INSERT INTO offers_setup (id, cluster_id, title, brand, category,
   ↪ description, price, identifiers, keyvaluepairs, spectablecontent,
   ↪ world_prob, attribute_prob)
    VALUES(-464, 77, ..., 0.42911, 0.629),
          (-466, 77, ..., 0.5, 0.246),
          (-468, 77, ..., 0.32635, 0.629),
          (-469, 77, ..., 0.5, 0.125),
          (-471, 77, ..., 0.24454, 0.629);

DROP TABLE IF EXISTS offers_rk_world CASCADE;
DROP TABLE IF EXISTS offers_rk_attrs CASCADE;
DROP TABLE IF EXISTS offers CASCADE;

CREATE TABLE offers_rk_world AS REPAIR KEY cluster_id IN offers_setup
   ↪ WEIGHT BY world_prob;
CREATE TABLE offers_rk_attrs AS REPAIR KEY id IN offers_setup WEIGHT BY
   ↪ attribute_prob;

CREATE TABLE offers AS (
    SELECT attrs.*
    FROM offers_rk_attrs AS attrs, offers_rk_world AS world
    WHERE attrs.id = world.id
);

-- Insert, Update, Delete 2:
INSERT INTO offers_setup (id, cluster_id, title, brand, category,
   ↪ description, price, identifiers, keyvaluepairs, spectablecontent,
   ↪ world_prob, attribute_prob)
SELECT * FROM bulk_insert;

DROP TABLE IF EXISTS offers_rk_world CASCADE;
DROP TABLE IF EXISTS offers_rk_attrs CASCADE;
DROP TABLE IF EXISTS offers CASCADE;

CREATE TABLE offers_rk_world AS REPAIR KEY cluster_id IN offers_setup
   ↪ WEIGHT BY world_prob;
CREATE TABLE offers_rk_attrs AS REPAIR KEY id IN offers_setup WEIGHT BY
   ↪ attribute_prob;

CREATE TABLE offers AS (
    SELECT attrs.*
    FROM offers_rk_attrs AS attrs, offers_rk_world AS world
    WHERE attrs.id = world.id
);

-- Insert, Update, Delete 3:
UPDATE offers
SET world_prob = 0.345,
    attribute_prob = 0.3992
WHERE _d0 = 615777
AND _d1 = 613619;

UPDATE offers
SET world_prob = 0.345,
    attribute_prob = 0.6008
WHERE _d0 = 615777
and _d1 = 613841;

UPDATE offers
SET world_prob = 0.1254,
    attribute_prob = 0.5
WHERE _d0 = 615999
AND _d1 = 613619;
```

```

UPDATE offers
SET world_prob = 0.1254,
    attribute_prob = 0.5
WHERE _d0 = 615999
and _d1 = 613841;

UPDATE offers
SET world_prob = 0.487,
    attribute_prob = 0.4300
WHERE _d0 = 615850
and _d1 = 613395;

UPDATE offers
SET world_prob = 0.487,
    attribute_prob = 0.5700
WHERE _d0 = 615850
AND _d1 = 613692;

UPDATE offers
SET world_prob = 0.487,
    attribute_prob = 0.4300
WHERE _d0 = 615999
and _d1 = 613841;

UPDATE offers
SET world_prob = 0.487
WHERE _d0 = 615553
AND _d1 = 613692;

UPDATE offers
SET world_prob = 0.487
WHERE _d0 = 615553
and _d1 = 613395;

UPDATE offers
SET world_prob = 0.329
WHERE _d0 = 614032
AND _d1 = 612733;

UPDATE offers
SET world_prob = 0.329
WHERE _d0 = 614032
and _d1 = 611874;

UPDATE offers
SET world_prob = 0.184
WHERE _d0 = 615197
AND _d1 = 612733;

UPDATE offers
SET world_prob = 0.184
WHERE _d0 = 615197
and _d1 = 613039;

DROP TABLE IF EXISTS offers_rk_world CASCADE;
```

```sql
DROP TABLE IF EXISTS offers_rk_attrs CASCADE;
DROP TABLE IF EXISTS offers CASCADE;

CREATE TABLE offers_rk_world AS REPAIR KEY cluster_id IN offers_setup WEIGHT BY world_prob;
CREATE TABLE offers_rk_attrs AS REPAIR KEY id IN offers_setup WEIGHT BY attribute_prob;

CREATE TABLE offers AS (
    SELECT attrs.*
    FROM offers_rk_attrs AS attrs, offers_rk_world AS world
    WHERE attrs.id = world.id
);

-- Insert, Update, Delete 4:
UPDATE offers_setup
SET cluster_id = max_cluster.max_id,
    world_prob = 1,
    attribute_prob = 1
FROM (
    SELECT max(cluster_id) + 1 AS max_id
    FROM offers_setup
) as max_cluster
WHERE id = 12071001;

UPDATE offers_setup
SET cluster_id = max_cluster.max_id,
    world_prob = 1,
    attribute_prob = 1
FROM (
    SELECT max(cluster_id) + 1 AS max_id
    FROM offers_setup
) as max_cluster
WHERE id = 16457529;

UPDATE offers_setup
SET world_prob = 0.63,
    attribute_prob = 0.5
WHERE id = 7339350;

UPDATE offers_setup
SET world_prob = 0.63,
    attribute_prob = 0.5
WHERE id = 12326926;

DROP TABLE IF EXISTS offers_rk_world CASCADE;
DROP TABLE IF EXISTS offers_rk_attrs CASCADE;
DROP TABLE IF EXISTS offers CASCADE;

CREATE TABLE offers_rk_world AS REPAIR KEY cluster_id IN offers_setup WEIGHT BY world_prob;
CREATE TABLE offers_rk_attrs AS REPAIR KEY id IN offers_setup WEIGHT BY attribute_prob;

CREATE TABLE offers AS (
    SELECT attrs.*
    FROM offers_rk_attrs AS attrs, offers_rk_world AS world
    WHERE attrs.id = world.id
);

-- Insert, Update, Delete 5:
DELETE FROM offers_setup
WHERE cluster_id = 41;

DROP TABLE IF EXISTS offers_rk_world CASCADE;
DROP TABLE IF EXISTS offers_rk_attrs CASCADE;
DROP TABLE IF EXISTS offers CASCADE;

CREATE TABLE offers_rk_world AS REPAIR KEY cluster_id IN offers_setup WEIGHT BY world_prob;
CREATE TABLE offers_rk_attrs AS REPAIR KEY id IN offers_setup WEIGHT BY attribute_prob;

CREATE TABLE offers AS (
    SELECT attrs.*
    FROM offers_rk_attrs AS attrs, offers_rk_world AS world
    WHERE attrs.id = world.id
);
```

B

WDC Product Offers Dataset

B.1. Elaboration on the Dataset

In order to analyse the data, an adaptation was made to the Python program developed by Flokstra, which was created for version 1.0 of the WDC product corpus dataset. The code used can be found at GitHub [30].

A selection of queries was ran to gain insights on the dataset. An interesting find is that there is a total of twenty five product categories and all offers have a category. Additionally, there is a total of eight identifiers, of which /sku and /productID are the most used.

B.2. JSON Structure of a Product Offer

To provide a clear example of the structure of the data, a product offer is displayed from which all information is known. Only about 0.5% of the product offers in the dataset contain no NULL values.

```
{
  "brand": "hp enterprise",
  "category": "Computers_and_Accessories",
  "cluster_id": 5481799,
  "description": "description ait1 35 70gb hot swap lvdpart number s
     ↪ option part 70 40375 03",
  "id": 519,
  "identifiers": [{
        "\/mpn": "[704037503]"
  }],
  "keyValuePairs": {
    "category": "proliant",
    "sub category": "tapedrive",
    "generation": "2 4gb",
    "part number": "142074 001",
    "products id": "12849",
    "tape type": "dat",
    "native capacity": "2gb",
    "interface type": "scsi",
    "compressed capacity": "4gb",
    "form factor": "5 25 inch",
    "": ""},
  "price": usd 651 95,
  "specTableContent": "specifications category proliant sub category tape drive generation 35 70gb part number 70 40375 03 products id 13255 capacity 35 70gb interface type scsi lvd enclosure type canister hot swap configuration type canister hot swap",
  "title": "null , hp 70 40375 03 ait1 35 gb hs lvd"
}
```

C

Performance

This appendix provides the performance measures of QuestionMark: The Dataset Generator when using different parameter values. The performance is measured over the blocking algorithm and matching algorithm separately. The provided times are the average time over the indicated amount of iterations. For information on the used methodology, see Section 4.3.

C.1. Blocking Algorithm Performance

Performance of the Adaptive Sorted Neighborhood blocking algorithm

Shows the runtime, precision and recall for varying values of the distance measure. See Figure C.1 for a visual representation of the data below.

```
time, precision, recall: dataset size,  distance,     parameters,              # iterations
 31 ms,    0.360, 0.239:  4 687 records, Levenshtein,  mbs = 30, phi = 0.4, ws = 6.  10 runs
 22 ms,    0.392, 0.521:  4 687 records, Jaro,         mbs = 30, phi = 0.4, ws = 6.  10 runs
 20 ms,    0.392, 0.521:  4 687 records, Jaro-Winkler, mbs = 30, phi = 0.4, ws = 6.  10 runs
436 ms,    0.496, 0.148:  4 687 records, Hamming,      mbs = 30, phi = 0.4, ws = 6.  10 runs
150 ms,    0.390, 0.367:  4 687 records, Jaccard,      mbs = 30, phi = 0.4, ws = 6.  10 runs
```

Shows the runtime, precision and recall for varying values of the distance threshold. See Figure C.2 for a visual representation of the data below.

```
time, precision, recall: dataset size,  distance,     parameters,              # iterations
 47 ms,    0.553, 0.109:  4 687 records, Jaro,         mbs = 30, phi = 0.1, ws = 2.  10 runs
 34 ms,    0.396, 0.403:  4 687 records, Jaro,         mbs = 30, phi = 0.3, ws = 2.  10 runs
 28 ms,    0.392, 0.512:  4 687 records, Jaro,         mbs = 30, phi = 0.4, ws = 2.  10 runs
 56 ms,    0.384, 0.512:  4 687 records, Jaro,         mbs = 30, phi = 0.5, ws = 2.  10 runs
```

Shows the runtime, precision and recall for varying values of the window size. See Figure C.3 for a visual representation of the data below.

```
time, precision, recall: dataset size,  distance,     parameters,              # iterations
 36 ms,    0.388, 0.496:  4 687 records, Jaro,         mbs = 30, phi = 0.4, ws = 1.  10 runs
 25 ms,    0.391, 0.517:  4 687 records, Jaro,         mbs = 30, phi = 0.4, ws = 3.  10 runs
 28 ms,    0.392, 0.521:  4 687 records, Jaro,         mbs = 30, phi = 0.4, ws = 6.  10 runs
 21 ms,    0.386, 0.519:  4 687 records, Jaro,         mbs = 30, phi = 0.4, ws = 10. 10 runs
```

Shows the runtime, precision and recall for varying values of the maximum block size. See Figure C.4 for a visual representation of the data below.

```
time, precision, recall: dataset size,   distance,     parameters,                # iterations
 33 ms,    0.458, 0.329:  4 687 records, Jaro,         mbs = 6,  phi = 0.4, ws = 6.  10 runs
 24 ms,    0.400, 0.471:  4 687 records, Jaro,         mbs = 20, phi = 0.4, ws = 6.  10 runs
 21 ms,    0.374, 0.563:  4 687 records, Jaro,         mbs = 50, phi = 0.4, ws = 6.  10 runs
 19 ms,    0.364, 0.570:  4 687 records, Jaro,         mbs = 80, phi = 0.4, ws = 6.  10 runs
```

Shows the runtime for varying values of the dataset size. See Figure C.5 for a visual representation of the data below.

```
time, precision, recall: dataset size,   distance,     parameters,                # iterations
 19 ms,       N/A:         1 171 records, Levenshtein, mbs = 30, phi = 0.2, ws = 2.  10 runs
 33 ms,       N/A:         2 343 records, Levenshtein, mbs = 30, phi = 0.2, ws = 2.  10 runs
410 ms,       N/A:        20 000 records, Levenshtein, mbs = 30, phi = 0.2, ws = 2.  10 runs
7532 ms,      N/A:        80 000 records, Levenshtein, mbs = 30, phi = 0.2, ws = 2.  10 runs
42778 ms,     N/A:       200 000 records, Levenshtein, mbs = 30, phi = 0.2, ws = 2.  5  runs
2276543 ms,   N/A:       999 000 records, Levenshtein, mbs = 30, phi = 0.2, ws = 2.  1  runs
```

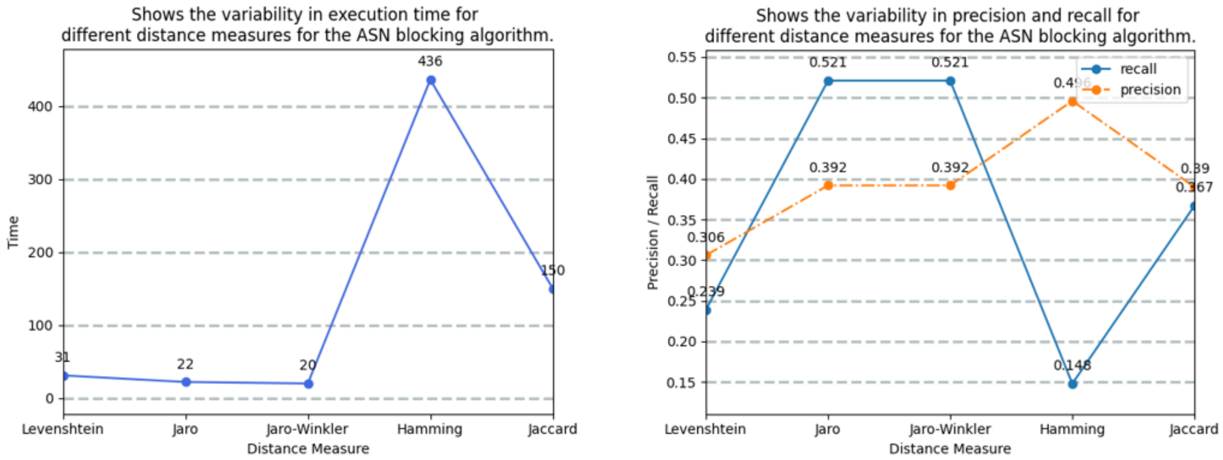


Figure C.1: Performance of the ASN algorithm varying distance measures.

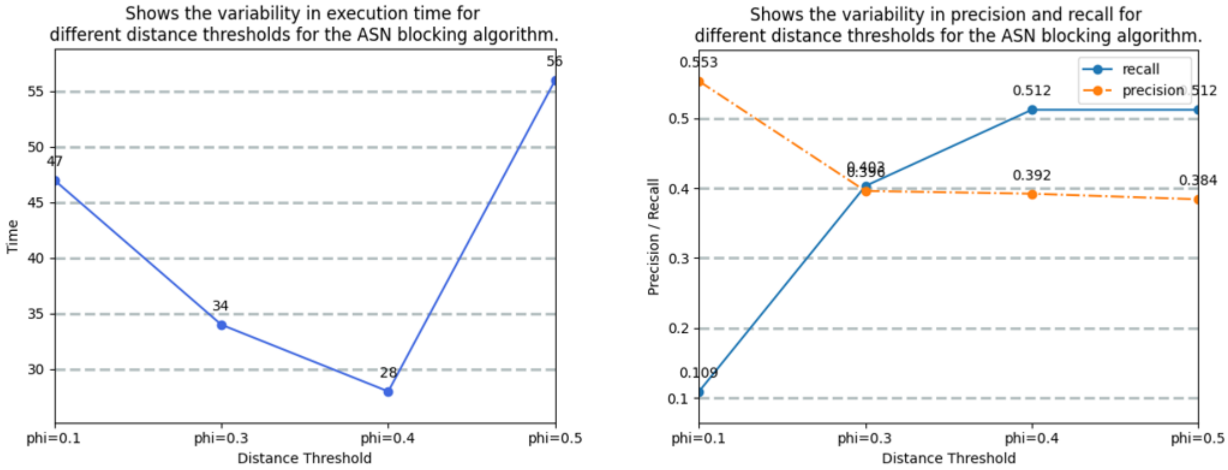


Figure C.2: Performance of the ASN algorithm for varying values of phi.

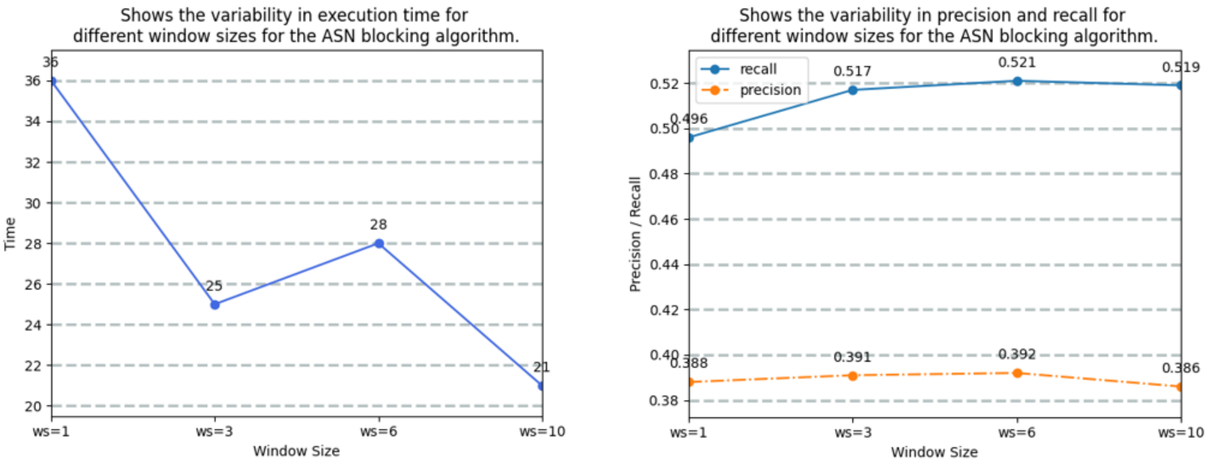


Figure C.3: Performance of the ASN algorithm for varying window sizes.

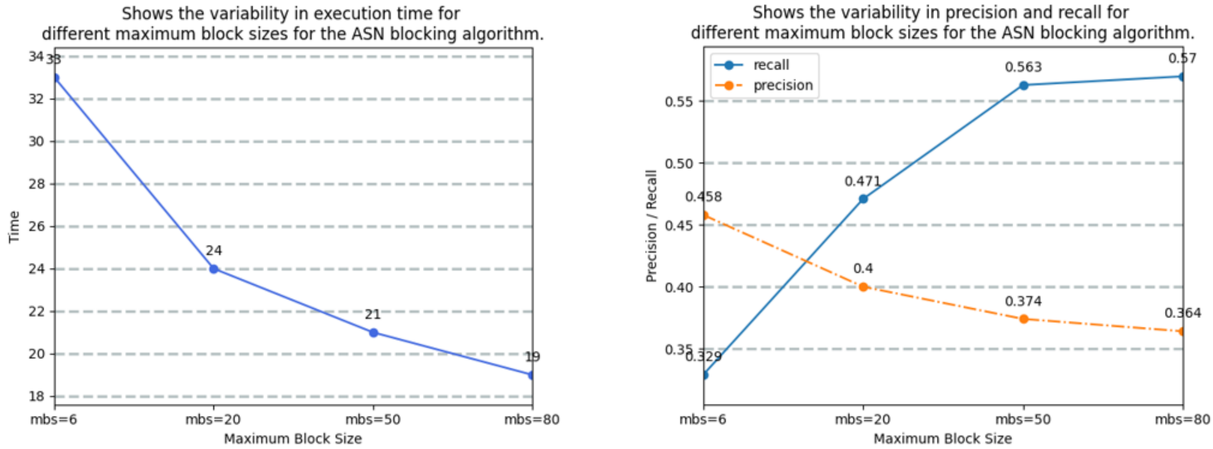


Figure C.4: Performance of the ASN algorithm for varying maximum block sizes.

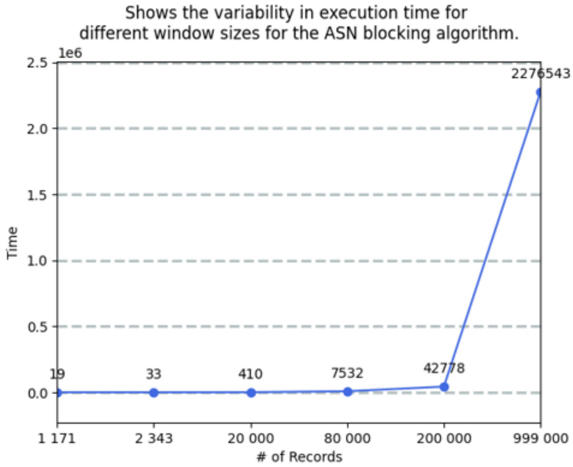


Figure C.5: Performance of the ASN algorithm for varying dataset sizes.

Performance of the Improved Suffix Array blocking algorithm

Shows the runtime, precision and recall for varying values of the distance threshold. See Figure C.6 for a visual representation of the data below.

```
time, precision, recall:  dataset size,  distance,      parameters,                     # iterations
1679 ms,   0.311, 0.425:   4 687 records, Jaro,          msl = 3, mbs = 30, phi = 0.2.  10 runs
1669 ms,   0.374, 0.845:   4 687 records, Jaro,          msl = 3, mbs = 30, phi = 0.3.  10 runs
1609 ms,   0.324, 0.994:   4 687 records, Jaro,          msl = 3, mbs = 30, phi = 0.4.  10 runs
 >3 min,       -,     -:   4 687 records, Jaro,          msl = 3, mbs = 30, phi = 0.5.  1  runs
```

Shows the runtime, precision and recall for varying values of the maximum block size. See Figure C.7 for a visual representation of the data below.

```
time, precision, recall: dataset size,  distance,      parameters,                   # iterations
1631 ms,   0.331, 0.993:   4 687 records, Jaro,          msl = 3, mbs = 6,  phi = 0.4.  10 runs
1695 ms,   0.329, 0.994:   4 687 records, Jaro,          msl = 3, mbs = 10, phi = 0.4.  10 runs
1560 ms,   0.328, 0.994:   4 687 records, Jaro,          msl = 3, mbs = 20, phi = 0.4.  10 runs
1593 ms,   0.322, 0.994:   4 687 records, Jaro,          msl = 3, mbs = 50, phi = 0.4.  10 runs
```

Shows the runtime, precision and recall for varying values of the minimum suffix length. See Figure C.8 for a visual representation of the data below.

```
time, precision, recall: dataset size,  distance,      parameters,                   # iterations
1616 ms,   0.324, 0.994:   4 687 records, Jaro,          msl = 1, mbs = 30, phi = 0.5.  10 runs
1565 ms,   0.324, 0.994:   4 687 records, Jaro,          msl = 5, mbs = 30, phi = 0.5.  10 runs
1386 ms,   0.330, 0.981:   4 687 records, Jaro,          msl = 15,mbs = 30, phi = 0.5.  10 runs
```

Shows the runtime, precision and recall for varying values of the distance threshold and distance measure.

```
time, precision, recall: dataset size,  distance,      parameters,                   # iterations
1339 ms,   0.267, 0.314:   4 687 records, Levenshtein,  msl = 3, mbs = 30, phi = 0.4.  10 runs
1609 ms,   0.292, 0.401:   4 687 records, Levenshtein,  msl = 3, mbs = 30, phi = 0.5.  10 runs
1316 ms,   0.320, 0.542:   4 687 records, Levenshtein,  msl = 3, mbs = 30, phi = 0.6.  10 runs
4051 ms,   0.278, 0.224:   4 687 records, Hamming,      msl = 3, mbs = 30, phi = 0.4.  10 runs
4167 ms,   0.310, 0.301:   4 687 records, Hamming,      msl = 3, mbs = 30, phi = 0.5.  10 runs
4031 ms,   0.330, 0.390:   4 687 records, Hamming,      msl = 3, mbs = 30, phi = 0.6.  10 runs
3929 ms,   0.347, 0.481:   4 687 records, Jaccard,      msl = 3, mbs = 30, phi = 0.4.  10 runs
4486 ms,   0.370, 0.698:   4 687 records, Jaccard,      msl = 3, mbs = 30, phi = 0.5.  10 runs
3993 ms,   0.359, 0.881:   4 687 records, Jaccard,      msl = 3, mbs = 30, phi = 0.6.  10 runs
1471 ms,   0.375, 0.800:   4 687 records, Jaro-Winkler, msl = 3, mbs = 30, phi = 0.3.  10 runs
1434 ms,   0.330, 0.981:   4 687 records, Jaro-Winkler, msl = 3, mbs = 30, phi = 0.4.  10 runs
```

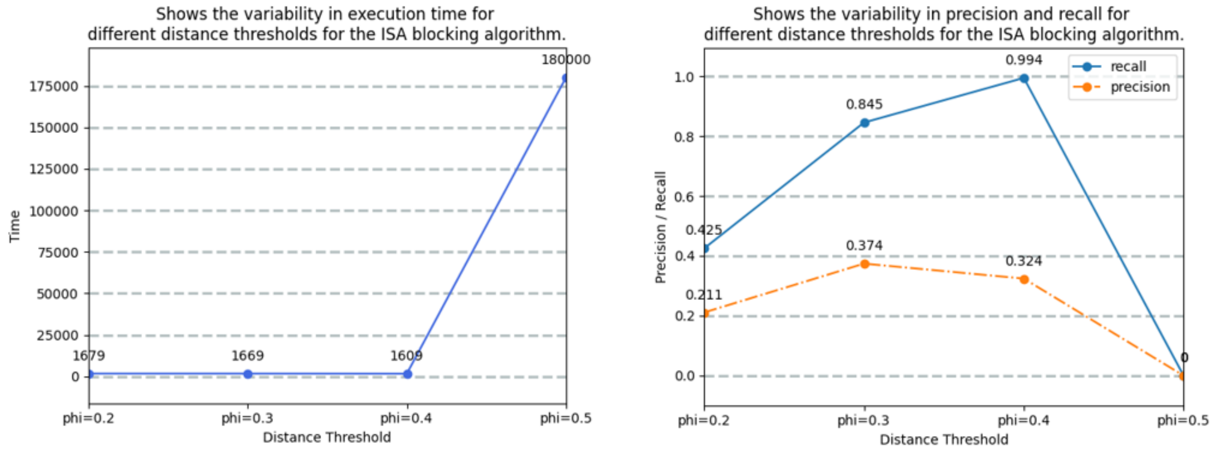


Figure C.6: Performance of the ISA algorithm for varying values of phi.

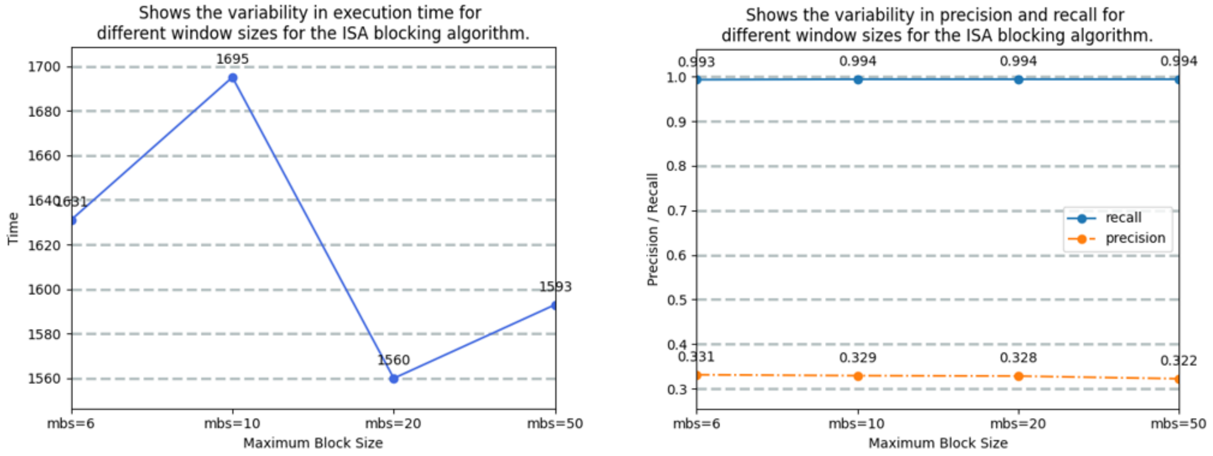


Figure C.7: Performance of the ISA algorithm for varying maximum block sizes.

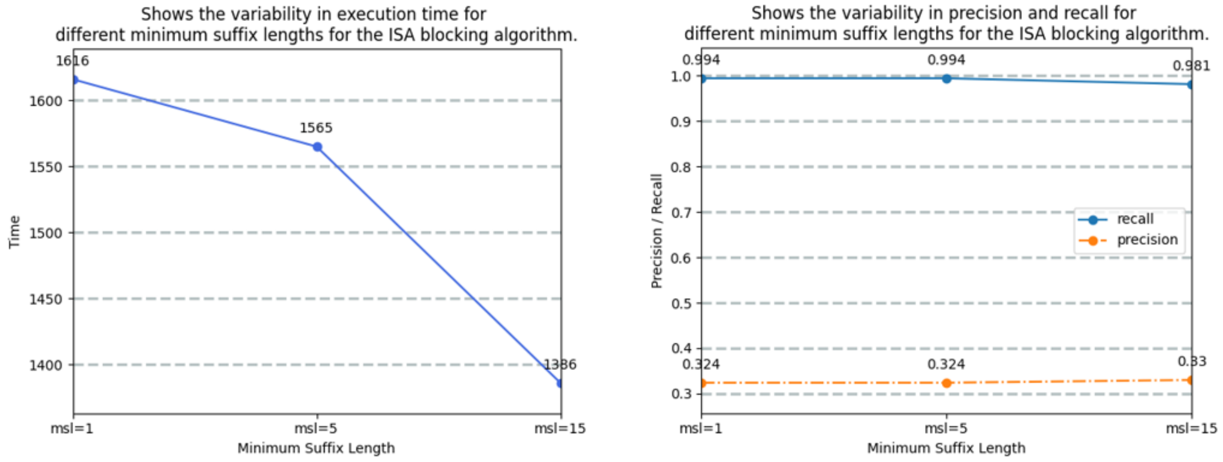


Figure C.8: Performance of the ISA algorithm for varying minimum suffix lengths.

C.2. Matching Algorithm Performance

For the matching algorithm, the attributes used for offer similarity are the ones that are in the default benchmark, which are brand, category, cluster_id, description, identifiers, keyValuePairs, price, specTableContent and title with weights 1, 0.7, 1, 0.8, 0.8, 0.8, 1, 0.7 and 1 respectively.

Performance of the Attribute-based Entity Resolution matching algorithm run on blocks created by Adaptive Sorted Neighborhood.

Shows the runtime, precision and recall for varying values of the lower distance threshold. Run on blocks created by ASN with parameters WS = 2, PHI = 0.36, MBS = 6. See Figure C.9 for a visual representation of the data below.

```
time,    exp.prec., exp.rec.:  dataset size, distance, parameters,                  # iterations
2588 ms,     0.073,     0.133: 4 687 records, Jaro,     low_phi = 0.08, upp_phi = 0.5.  10 runs
162 ms,      0.061,     0.313: 4 687 records, Jaro,     low_phi = 0.12, upp_phi = 0.5.  10 runs
115 ms,      0.061,     0.349: 4 687 records, Jaro,     low_phi = 0.15, upp_phi = 0.5.  10 runs
117 ms,      0.060,     0.358: 4 687 records, Jaro,     low_phi = 0.20, upp_phi = 0.5.  10 runs
114 ms,      0.060,     0.358: 4 687 records, Jaro,     low_phi = 0.30, upp_phi = 0.5.  10 runs
```

Shows the runtime, precision and recall for varying values of the upper distance threshold. Run on blocks created by ASN with parameters WS = 2, PHI = 0.36, MBS = 6. See Figure C.10 for a visual representation of the data below.

```
time,    exp.prec., exp.rec.:  dataset size, distance, parameters,                  # iterations
114 ms,      0.060,     0.358: 4 687 records, Jaro,     low_phi = 0.20, upp_phi = 0.22. 10 runs
114 ms,      0.061,     0.313: 4 687 records, Jaro,     low_phi = 0.20, upp_phi = 0.3.  10 runs
114 ms,      0.060,     0.358: 4 687 records, Jaro,     low_phi = 0.20, upp_phi = 0.7.  10 runs
```

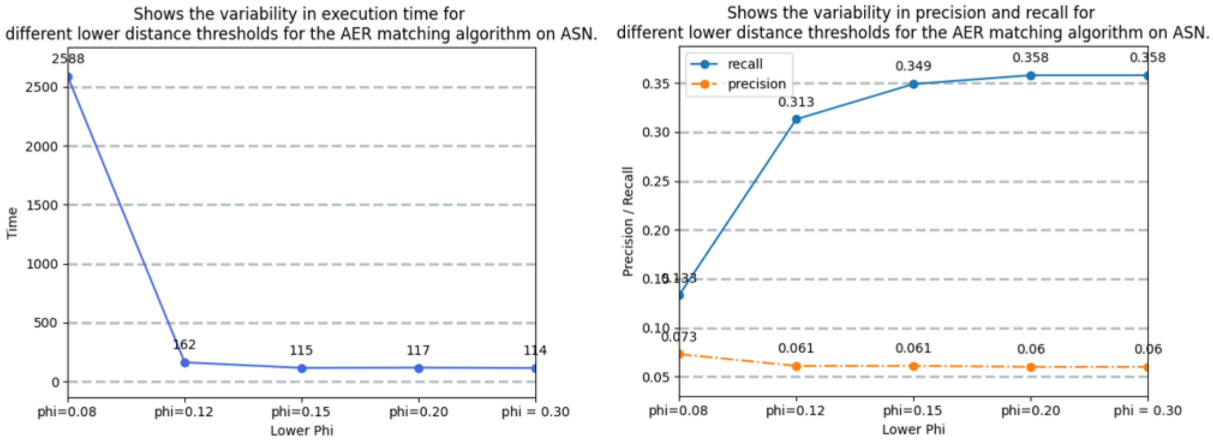


Figure C.9: Performance of the AER algorithm on ASN for varying values of the lower phi.

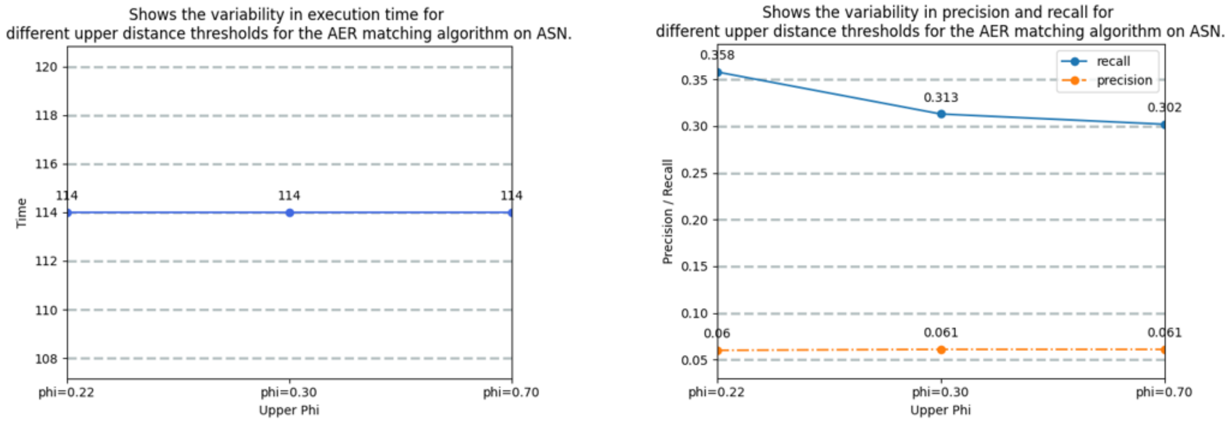


Figure C.10: Performance of the AER algorithm on ASN for varying values of the upper phi.

Performance of the Attribute-based Entity Resolution algorithm run on blocks created by Improved Suffix Array.

Shows the runtime, precision and recall for varying values of the upper and lower distance threshold. Run on blocks created by ISA with parameters PHI = 0.36, MBS = 6, MSL = 3. See Figure C.11 for a visual representation of the data below.

```
time,    exp.prec., exp.rec.:   dataset size, distance, parameters,                  # iterations
 > 5 min,         -,         -: 4 687 records, Jaro,     low_phi = 0.22, upp_phi = 0.25.  1 run
21104 ms,    0.005,     0.978: 4 687 records, Jaro,     low_phi = 0.25, upp_phi = 0.30. 10 runs
21594 ms,    0.005,     0.977: 4 687 records, Jaro,     low_phi = 0.30, upp_phi = 0.35.  5 runs
```

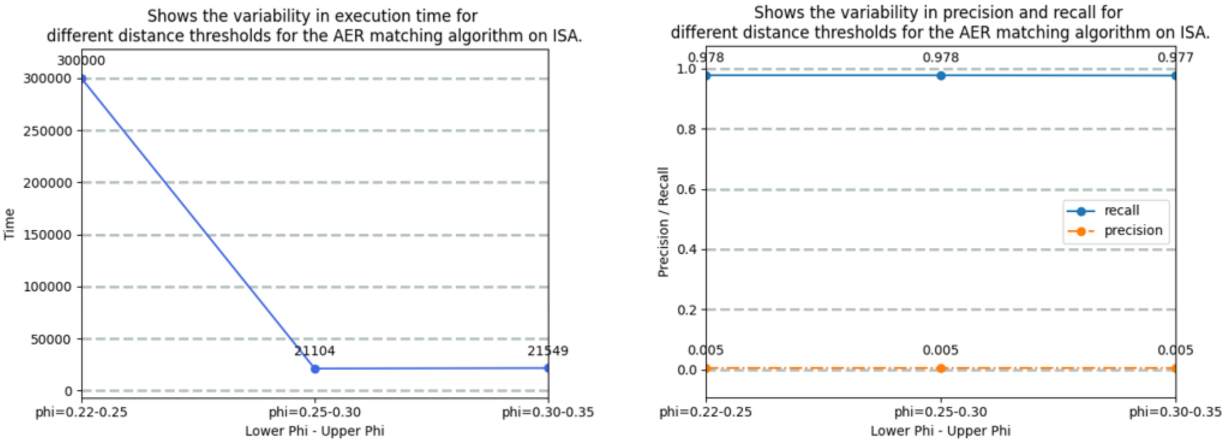


Figure C.11: Performance of the AER algorithm on ISA for varying values of phi.

D
User Study

This appendix provides additional information on the user study performed in this research. The user study validates the design of the benchmark and was used to improve the design of the benchmark. In this appendix, the informed consent form that was shared with all participants can be found, as well as the task provided for the formal experiment and the questions asked during the interview.

D.1. Informed Consent Form

Before conducting the research, the participants were informed on the research purpose and its potential risks. This was done both written and verbally. The written form is included as it was provided to the participants on paper.

Informed Consent Form
QuestionMark User Study

Project title
The design of the QuestionMark probabilistic benchmark.

Purpose of the study
I am inviting you to participate in this research project about the QuestionMark benchmark. This research is being conducted to obtain insights in the usability of the designed benchmark and to improve its design.

Procedures
You will participate in a formal experiment lasting approximately one hour. You will be asked to install and run the designed benchmark by the instructions provided by the program itself. This part should take 30 to 45 minutes. When completed, you will be asked questions on your experiences working with the benchmark and on possible points of improvement.

Potential risks and discomforts
There are no obvious physical, legal or economic risks associated with participating in this study. You do not have to answer any questions you do not wish to answer. Your participation is voluntary and you are free to discontinue your participation at any time.

Potential benefits
Participation in this study does not guarantee any beneficial results to you. As a result of participating you may better understand the concept of benchmarking and probabilistic databases. The broader goal of this research is to deliver a benchmark so that current and novel probabilistic database management systems can be benchmarked on performance, and to stimulate the use of these systems in day-to-day business processes.

Confidentiality
Your privacy will be protected to the maximum extent possible. No personally identifiable information will be reported in any research product. Only the researcher will have access to your responses. Within these restrictions, results of this study will be made available to you upon request. At the start of your research your name will be replaced by a pseudonym; your name will be coded.

The data will be stored in the University of Twente OneDrive. The original data will be deleted four weeks after the experiment or when the paper is published, whichever is soonest.

Compensation
You will not be compensated for participation in this research.

Right to withdraw and questions

Your participation in this research is completely voluntary. You may choose not to take part at all. If you decide to participate in this research, you may stop participating at any time. If you decide not to participate in this study or if you stop participating at any time, you will not be penalized or lose any benefits to which you otherwise qualify. You do not need to provide any explanation on why you stop participating. After the experiment you have five working days to withdraw your consent; after that your responses have been included in the anonymized experiment results and cannot be traced to you anymore. If you provide consent, the data you provided before you stopped participating will be processed in this research; no new data will be collected or used.

If you decide to stop taking part in the study, if you have questions, concerns, or complaints, or if you need to report an injury related to the research, please contact the primary investigator:

Nikki Zandbergen
n.zandbergen@student.utwente.nl

Statement of consent

Your signature indicates that you are at least 16 years of age; you have read this consent form or have had it read to you; your questions have been answered to your satisfaction and you voluntarily agree that you will participate in this research study. You will receive a copy of this signed consent form.

I agree to participate in a research project led by Nikki Zandbergen. The purpose of this document is to specify the terms of my participation in the project through being interviewed.

1. I have been given sufficient information about this research project. The purpose of my participation as an interviewee in this project has been explained to me and is clear.

2. My participation as an interviewee in this project is voluntary. There is no explicit or implicit coercion whatsoever to participate.

3. Participation involves following a formal experiment followed by getting interviewed by the main researcher. This experiment will last 60 minutes. I allow the researcher to take written notes during the interview. It is clear to me that in case I do not want my interview answers to be written down I am at any point of time fully entitled to withdraw from participation.

4. I have the right not to answer any of the questions. If I feel uncomfortable in any way during the interview session, I have the right to withdraw from the interview.

5. I have been given the explicit guarantees that the researcher will not identify me by name or function in any reports using information obtained from this interview, and that my confidentiality as a participant in this study will remain secure.

6. I have been given the guarantee that this research project has been reviewed and approved by the Ethics Committee of Computer and Information Science of the University of Twente. For research problems or any other question regarding the research project, the Secretary of the Ethics Committee at University Twente may be contacted through ethicscommittee-cis@utwente.nl.

7. I have read and understood the points and statements of this form. I have had all my questions answered to my satisfaction, and I voluntarily agree to participate in this study.

8. I have been given a copy of this consent form co-signed by the interviewer.

________________________ ________________________ _________

Participant Signature Date

________________________ ________________________ _________

Researcher Signature Date

D.2. Formal Experiment

The first part of the user study consisted of a formal experiment. Before the start of this part, the following information was read to the participants:

> Thank you for participating in my research. You have been informed on the purpose of this research and its risks. Have you signed the informed consent form? You can ask questions at any point or quit the research without the need of stating a reason. Can you confirm again whether you want to participate in this research?
>
> So this research will take about an hour. It is a long sit, so if you would like anything to drink or have a short break please let me know.
>
> For the first half, I will ask you to install and run the benchmark as if you were an employee of a company that is asked to benchmark their software. This means that you only have the information provided by the benchmark at hand. The database is already set up, so you only need to establish the connection to it.
>
> For the second part, I will ask you some questions regarding your experiences with the benchmark.
>
> If you have any questions or remarks about the benchmark you may ask or state them directly as they pop into your head.
>
> Do you have any questions before we start?

The participants were then provided with the following information on paper:

> On the laptop given to you there is a browser with the GitLab webpage of the benchmark already opened.
>
> There is also a folder ‘UserStudy’ already opened. You can use this folder to save anything you deem necessary. During the process, you need to zip some created files. Check the back of this paper for instructions.
>
> Good luck! You can always ask questions or take a break.
>
> You need a smaller dataset to run the benchmark. You find out that the parameters are already set correctly for the requirements of your company.
>
> Your company has the following database credentials:

```
host=127.0.0.1
database=postgres
user=postgres
password=postgres
port=5433
```

D.3. Interview Questions

After the participant completed the formal experiment, a small interview was conducted. For this interview, the following questions were asked.

1. What do you think?
2. How would you rate the user manual provided with the benchmark? Did you have trouble understanding specific steps? Were steps missing?
3. How do you rate the user-friendliness? Was it easy or difficult to use?
4. How would you rate your own Python level? Was that sufficient to run the benchmark?
5. How much previous knowledge on benchmarking and (probabilistic) database technology do you have?
6. Can you understand the provided results?
7. Do you have any further suggestions for improvement?
8. Anything else you would like to add?

D.4. Extensive Results

The following feedback was collected during each of the user studies. Before each item, the type of feedback is indicated between square brackets. Here, `[o]` indicates that the feedback was collected by means of observation by the researcher and `[c]` indicates that it was a comment provided by the subject. Answers on questions are indicated by `[q1]` etc.

Participant #1

`[c]` It is unclear what Python version is required. Does any version work?

`[o]` It was not directly clear that the project should be downloaded from GitLab.

`[o]` The laptop used to test the benchmark on posed some struggles to set up a working Python environment to download and run the benchmark in.

`[o]` Indicate that for the dataset download the normalized version is needed.

`[c]` It is unclear what parameters are required for the setup.

`[c]` Is is unclear whether a smaller dataset should be generated.

`[c]` It feels complicated that the functions required to run QuestionMark: The Dataset Generator should be manually uncommented and run. This should be automated.

`[c]` It should be indicated more clearly whether the size parameter uses a percentage or a percentile.

`[o]` Instructions should be added to the manual on how to zip the produced files.

`[c]` The manual is a bit wordy. Several steps can be combined and the instructions on the ISA algorithm cause confusion.

[c] For the software to work, you need to make too many clicks. First of all to explore the structure of the software, but also to actually run the software.

[o] The port was not included in `database.ini.tmpl`. This should be added.

[o] It was not clear that the performance tests of QuestionMark: The Dataset Generator should not be run for the benchmarking procedure.

[c] The manual does not state clearly enough that once the dataset is produced you need to continue to QuestionMark: The Probabilistic Benchmark.

[c] Also in QuestionMark: The Probabilistic Benchmark the process should be more automated.

[c] In the benchmark results document it would be clearer if the query time would be displayed above the query result.

[q1] Since the participant had no prior knowledge on database benchmarking it would be appreciated if more explanation is included on what processes are included and why they are of importance.

[q2] The user manual had some errors. These errors were all mentioned during the study. It is advised to provide users the link to QuestionMark: The Probabilistic Benchmark and let them start there, since that is the main program.

[q4] The Python level was sufficient to run the benchmark test.

[q6] The provided results were understandable, but due to a lack of knowledge on the subject it is difficult to turn it into insights.

[q7] All suggestions have been made during the study.

[q8] No further comments.

Participant #2

[o] It was not directly clear that the project should be downloaded from GitLab.

[o] There were again problems with the Python environment on the provided laptop.

[c] The project is difficult to navigate. It requires too many clicks.

[c] It should be explained more clearly on how the functioning database should be set up, in case that process is not performed by a database administrator.

[c] The percentages that are displayed during runtime are nice.

[c] The numbering in the manual is incorrect. It jumps from 4 to 6.

[o] It was not clear that the performance tests of QuestionMark: The Dataset Generator should not be run for the benchmarking procedure.

[c] The structure of QuestionMark: The Probabilistic Benchmark feels different than that of QuestionMark: The Dataset Generator. The structure of both should be similar.

[q1] The benchmark itself is clear, but the indicated points should be improved.

[q2] The participant is content with the manual. It was clear and contained elaborate instructions. Explanation on why some steps are required is lacking and should be implemented.

[q4] The participant had basic Python knowledge but deemed that sufficient.

[q5] The participant had no prior knowledge on benchmarking and of probabilistic database technology.

[q6] The overview provided by the results was clear.

[q7] Apart from the already provided feedback there are no further comments.

[q8] No further comments.

Participant #3

[c] The manual should state more clearly that a dataset needs to be generated.

[c] Make the reference to `MANUAL.md` within `README.md` a clickable link.

[c] Add additional explanation to the manual. You can hide the additional explanation with markdown ensuring that it doesn't clutter the page.

[c] It is annoying that the program requires a lot of clicks before you can actually start the dataset generation and benchmarking process.

[c] The participant was not sure whether they would usually follow the manual step-by-step or whether they would scan the product quickly and work out the functioning based on their own expertise. The participant mentioned that the manual reminded them of an IKEA-manual.

[c] The manual contains various typos.

[c] There should be a clearer explanation on the difference between QuestionMark: The Dataset Generator and QuestionMark: The Probabilistic Benchmark.

[c] The manual should indicate that the downloaded dataset should be included unzipped within the created dataset folder.

[c] The link to the WDC dataset website should be https instead of http.

[c] The overview on the performance of QuestionMark: The Dataset Generator should contain several graphs to display the information more clearly.

[c] The information on how to include a new DBMS should be included in a drop down text box, instead of its current location at the bottom.

[o] The manual zipping process should also be automated.

[c] The display of the progress should also be included in the resize dataset process. It would also be better to show the process with a progress bar instead of printing the percentage.

[c] The manual should provide more explanation on the included processes and why they are performed. What does the dataset generator do? Why is it needed? What are offers? Explain that it makes the downloaded dataset probabilistic. Include more technical details on the functioning of the program.

[c] Both programs should provide a clearer finished message.

[c] Include in QuestionMark: The Probabilistic Benchmark why it is required to also use QuestionMark: The Dataset Generator.

[c] When reaching the step to check the settings in `parameters.py`, walk the participants through the functioning of the parameters.

[c] Include a table with the included parameters and their functioning in `MANUAL.md`.

[c] Put the query explanations next to the query names in `parameters.py`.

[c] It is great that the benchmark contains sufficient good quality explanation and documentation.

[c] The results in the metrics result file requires more explanation.

[c] In QuestionMark: The Probabilistic Benchmark, include a list with common errors and how to fix them.

[c] If this experiment is a test on whether a monkey can follow a manual, then you succeeded.

[q1] QuestionMark is a nice tool. It is properly set up, also with the documentation. The participant was still as clueless about probabilistic databases as when they began the study. Due to a lack of knowledge they could not really tell more about the quality of the tooling. That felt like a shame.

[q2] The fact that there were four different markdown files was a bit annoying, that automatically requires a lot of clicks.

[q3] The manual covered all necessary steps and is dummy-proof.

[q4] The participant did not have much experience with programming in Python, but that knowledge was sufficient to work with the tool.

[q5] No knowledge on benchmarking or database technology.

[q6] The results were hard to understand. Especially the results with percentages require additional explanation.

[q7] Some additional explanation on Docker and setting up the database environment would be good. Another metric that could be included is the expenses required to run the benchmark test in terms of energy usage.

[q8] It would be nice to have a ten minute video on how to use the benchmark.

Participant #4

[c] The information displayed in `performance.txt` was not instantly clear. The information from `database.ini.tmpl` was clear.

[o] The participant could follow the steps and did not struggle much during the benchmarking process. Additional explanation about the process was required.

[q1] The participant wanted to know why the use of probabilistic database technology is interesting. As the end user is not by definition an expert on the subject additional explanation on the technology and why it is beneficial to use it is desired.

[q2] The manual was clear. It should be indicated where `database.ini` should be put. Additional information on how to set up a functioning database connection is also desired.

[q3] The software was easy to use, but additional explanation was required.

[q4] The participant was proficient with Python. The level was sufficient to run the benchmark.

[q5] The participant had no previous knowledge on benchmarking or probabilistic database technology.

[q6] The participant can understand the provided results, but does not know the meaning of the results yet. Additional explanation is desired.

[q7] No further suggestions.

[q8] No further comments.

Participant #5

[o] The participant had no trouble understanding that the project should be downloaded from GitLab to use it.

[o] The participant took some time to read through the instructions. The participant did not look at the explanation on what the benchmark does.

[o] The participant struggled to find the required files in the directory overview.

[o] The participant had no trouble understanding the instructions regarding the creation of `database.ini`.

[o] The participant could run QuestionMark: The Dataset Generator independently.

[c] The instructions regarding the database connection test in QuestionMark: The Probabilistic Benchmark are unclear.

[q1] The participant thought the program was nice. The participant would like to see more information on the required dependencies to run the program. What Python version is required? What PostgreSQL version is required?

[q2] The introduction was clear and it was nice that all program parts had the same structure. It was clear how the program should be run, but more information on what it does is desired. The participant also wanted more information on the dataset.

[q3] It was easy to use. Python knowledge is not really required. But for proper use, more knowledge on the subject is required and more explanation on the code is desired.

[q4] The participant is advanced in programming and also knows Python well. The level was more than sufficient.

[q5] The participant had knowledge on databases and had worked with benchmarking on a different area of research.

[q6] The provided results were understandable.

[q7] Due to the previous experiences with benchmarking tools, the participant thought that QuestionMark: The Probabilistic Benchmark would generate complex queries that the user could tailor to fit their needs. The participant also suggested to additionally generate the results in a JSON-format, so that the results could also be read by machines.

[q8] No further comments.

Participant #6

[c] The README feels fancy.

[o] The participant takes time to read the instructions.

[o] The participant noticed that the project had to be cloned before it could be used.

[o] The participant had no trouble finding the dataset on the WDC website.

[c] There are some inconsistencies within both projects regarding the name of files and their location. This should be fixed.

[c] The required specifications for the project should be mentioned somewhere. What Python version is required? Are there external dependencies? How should you set up your virtual environment? Pip freeze can be used to obtain a requirements document.

[c] In both systems, the instruction to set up a working database connection is unclear. It should state something in the lines of: 'Make sure the database management system of choice is running and is ready to accept connections'.

[c] In QuestionMark: The Probabilistic Benchmark, step 2 contains too many substeps. For clarity, they should each have their own step.

[c] In `parameters.py`, put TEST on top of the document, as that is probably the parameter that will be changed the most often.

[o] The participant wanted to know whether the program could be run from the command line, which worked.

[c] When running the test, the message 'if no errors are shown then it works' feels a bit like a make do solution. There should be a better solution for that.

[c] The error messages thrown during the benchmark execution are slightly confusing. It feels like something significant crashes and it is not directly clear that it is part of the benchmark.

[c] The results in a .txt file are not automatable. You would like to have them in either a CSV or JSON format. It also needs to be machine readable.

[c] The structure of `QM_metric_results` is also not directly clear. It is a lot of text.

[c] Instead of displaying the character count, there should be a standardised manner for displaying the complexity of a query.

[c] In the graph containing the runtimes, the orange bars displaying the planning times are very insignificant. It might be better to omit them.

[c] The results in `QM_query_results.txt` are not directly understandable. It is also due to a lack of understanding of what the results are meant to say.

[q1] It was mostly following the manual. The parameter files are nice; you can alter values centrally, it is highly configurable while still having the most used values already included. The reference to both programs in the manual are clear. It is understandable what should be done and the workflow is clear.

[q2] See the provided feedback. What was written was clear.

[q3] It was easy to use. It was nice that no command line arguments were included. It is really *run and go*.

[q4] Very sufficient in Python. Participant commented they are overqualified.

[q5] The participant has slight knowledge on benchmarking and has a more extensive database knowledge due to their study background.

[q6] The participant does not understand what information should be extracted from the provided results. They were shown the extensive user manual, which made the process of interpreting the results more clear for them.

[q7] Nothing more than what was already indicated.

[q8] No further comments.

E

Additional Details Case Study

This appendix provides additional information on the test setup and provides the raw results for MayBMS and DuBio. The interpretation of these results can be found in Section 6.1.

E.1. Docker images

Both MayBMS and DuBio were run in Docker. The below Docker images were used to build a container with the DBMS. The hardware is a single machine with AMD Ryzen 5 PRO 6650U CPU @ 2.90 GHz, quad core, 16 GB RAM, and 512 GB SSD PCIe Gen4.

Docker image for DuBio

```
FROM postgres:latest

RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt install -y \
    git \
    make \
    gcc \
    postgresql-server-dev-all \
    && rm -rf /var/lib/apt/lists/*

RUN git clone https://github.com/utwente-db/DuBio.git

WORKDIR /DuBio/pgbdd

RUN make install
```

Docker image for MayBMS

```
FROM debian:buster-slim

ENV PGDATA /usr/local/pgsql/data

RUN apt-get update && apt-get install -y \
    git \
    make \
    gcc \
    libreadline-dev \
    zlib1g-dev \
    bison \
    flex \
    && rm -rf /var/lib/apt/lists/*

RUN git clone https://git.code.sf.net/p/maybms/code maybms

# COPY maybms-src maybms

WORKDIR /maybms/postgresql-8.3.3

RUN ./configure CFLAGS=-fno-aggressive-loop-optimizations

RUN make

RUN make install

RUN adduser postgres

WORKDIR /usr/local/pgsql

RUN mkdir ./data && chown postgres ./data

USER postgres

RUN ./bin/initdb -D ./data

RUN sed -i "/^#listen_addresses/c\\listen_addresses = '*'" ./data/postgresql.conf && \
    echo "host  all all 0.0.0.0/0   md5" >> ./data/pg_hba.conf

EXPOSE 5432

CMD ["./bin/postgres", "-c", "config_file=./data/postgresql.conf"]
```

E.2. Raw Results

This appendix contains the results as produced by QuestionMark: The Probabilistic Benchmark.

The following results are included, in order of appearance:

- The metrics results file based on MayBMS queries.
- The metrics results file based on MayBMS statements.
- The metrics results file based on DuBio queries.
- The metrics results file based on DuBio statements.
- The query results file based on all MayBMS queries.
- The query results file based on all MayBMS statements.
- The query results file based on all DuBio queries
- The query results file based on all DuBio statements.

```
      # ============================================= #
      # ==============   QuestionMark  ============== #
      # ============================================= #
      # The metrics file.
      # Run on MayBMS.

 This file contains the metrics of this benchmark test.
 Please see 'QM_query_results.txt' for the results and runtimes of the queries.

 The included metrics produced the following results:
 +---------------------------------------------------------------------------+-------------------------+
 |        METRIC                                                             |       VALUE             |
 +---------------------------------------------------------------------------+-------------------------+
 | The total size of the probability space is:                               | 3240 kB                 |
 +---------------------------------------------------------------------------+-------------------------+
 | The total size of the duplicate records is:                               | 875.5 kB                |
 +---------------------------------------------------------------------------+-------------------------+
 | The percentage of data used for probabilistic representation:             | 80.39%                  |
 +---------------------------------------------------------------------------+-------------------------+
 | The total number of characters needed for all queries:                    | 1412 characters         |
 +---------------------------------------------------------------------------+-------------------------+
 | The percentage of successful queries is:                                  | 92.31%                  |
 |   (See below what functionality might be lacking)                         |                         |
 +---------------------------------------------------------------------------+-------------------------+
 | The total runtime of all queries is:                                      | 122.76 ms               |
 |   (The sum of all time averages over 5 iterations)                        |                         |
 +---------------------------------------------------------------------------+-------------------------+


 # An overview of the queries that finished with their execution time:
 +-----------------+------+---------+---------------+----------------+
 |   QUERY NAME    | Done | Runtime | Planning Time | Execution Time |
 +-----------------+------+---------+---------------+----------------+
 | test_1          |  x   | 0.031   | -             | -              |
 +-----------------+------+---------+---------------+----------------+
 | insight_1       |  x   | 1.521   | -             | -              |
 +-----------------+------+---------+---------------+----------------+
 | insight_2       |  x   | 2.272   | -             | -              |
 +-----------------+------+---------+---------------+----------------+
 | insight_3       |  x   | 4.039   | -             | -              |
 +-----------------+------+---------+---------------+----------------+
 | insight_4       |  x   | 9.817   | -             | -              |
 +-----------------+------+---------+---------------+----------------+
 | insight_5       |  x   | 3.624   | -             | -              |
 +-----------------+------+---------+---------------+----------------+
 | insight_6       |  x   | 9.11    | -             | -              |
 +-----------------+------+---------+---------------+----------------+
 | probabilistic_1 |  x   | 49.65   | -             | -              |
 +-----------------+------+---------+---------------+----------------+
 | probabilistic_2 |  x   | 11.338  | -             | -              |
 +-----------------+------+---------+---------------+----------------+
 | probabilistic_3 |  x   | 14.41   | -             | -              |
 +-----------------+------+---------+---------------+----------------+
 | probabilistic_4 |      | -       | -             | -              |
 +-----------------+------+---------+---------------+----------------+
 | probabilistic_5 |  x   | 13.316  | -             | -              |
 +-----------------+------+---------+---------------+----------------+
 | probabilistic_6 |  x   | 3.637   | -             | -              |
 +-----------------+------+---------+---------------+----------------+



 # ==== Overview of errors and possible missing functionality. ==== #
 # Based on the errors thrown during benchmark testing, the batabase system might
 # lack support for one or more functionalities. Please also verify the actual error
 # thrown by the database manually, as that may provide a clearer indication of what
 # went wrong during benchmarking. If a memory allocation error is thrown, you can
 # alter the query to run it on the 'part' table to see if the functionality of the
 # query is supported.

 # The following queries have raised an error:

 QUERY #:                probabilistic_4
 Functionality message:  The DBMS might lack the ability to calculate the probability of a composed query result.
                        It may struggle composing large and/or complex sentences/probability spaces.
 Error raised by DBMS:
column "variables" does not exist
LINE 2:     SELECT category, conf() AS probability, variables
                                                    ^
```

```
     # ============================================= #
     # ==============   QuestionMark  ============== #
     # ============================================= #
     # The metrics file.
     # Run on MayBMS.

This file contains the metrics of this benchmark test.
Please see 'QM_query_results.txt' for the results and runtimes of the queries.

The included metrics produced the following results:
+---------------------------------------------------------------------------+-------------------------+
|        METRIC                                                             |       VALUE             |
+---------------------------------------------------------------------------+-------------------------+
| The total size of the probability space is:                               | 3240 kB                 |
+---------------------------------------------------------------------------+-------------------------+
| The total size of the duplicate records is:                               | 875.5 kB                |
+---------------------------------------------------------------------------+-------------------------+
| The percentage of data used for probabilistic representation:            | 80.39%                  |
+---------------------------------------------------------------------------+-------------------------+
| The total number of characters needed for all queries:                    | 1761 characters         |
+---------------------------------------------------------------------------+-------------------------+
| The percentage of successful queries is:                                  | 100.0%                  |
|   (See below what functionality might be lacking)                         |                         |
+---------------------------------------------------------------------------+-------------------------+
| The total runtime of all queries is:                                      | 111.05 ms               |
|   (The sum of all time averages over 5 iterations)                        |                         |
+---------------------------------------------------------------------------+-------------------------+


# An overview of the queries that finished with their execution time:
+-----------------+------+---------+---------------+----------------+
|   QUERY NAME    | Done | Runtime | Planning Time | Execution Time |
+-----------------+------+---------+---------------+----------------+
| IUD_1_rollback  |  x   | 0.569   | -             | -              |
+-----------------+------+---------+---------------+----------------+
| IUD_2_rollback  |  x   | 106.54  | -             | -              |
+-----------------+------+---------+---------------+----------------+
| IUD_3_rollback  |  x   | 2.56    | -             | -              |
+-----------------+------+---------+---------------+----------------+
| IUD_4_rollback  |  x   | 0.68    | -             | -              |
+-----------------+------+---------+---------------+----------------+
| IUD_5_rollback  |  x   | 0.703   | -             | -              |
+-----------------+------+---------+---------------+----------------+
      # ============================================= #
      # ==============   QuestionMark  ============== #
      # ============================================= #
      # The metrics file.
      # Run on DuBio.

 This file contains the metrics of this benchmark test.
 Please see 'QM_query_results.txt' for the results and runtimes of the queries.

 The included metrics produced the following results:
 +---------------------------------------------------------------------------+-------------------------+
 |        METRIC                                                             |        VALUE            |
 +---------------------------------------------------------------------------+-------------------------+
 | The total size of the _sentence column is:                                | 160 kB                  |
 +---------------------------------------------------------------------------+-------------------------+
 | The total size of the dict table is:                                      | 784 kB                  |
 +---------------------------------------------------------------------------+-------------------------+
 | The percentage of data used for probabilistic representation:             | 50.64%                  |
 +---------------------------------------------------------------------------+-------------------------+
 | The total number of characters needed for all queries:                    | 1846 characters         |
 +---------------------------------------------------------------------------+-------------------------+
 | The percentage of successful queries is:                                  | 92.31%                  |
 |   (See below what functionality might be lacking)                         |                         |
 +---------------------------------------------------------------------------+-------------------------+
 | The total planning time of all queries is:                                | 1.64 ms                 |
 |   (The sum of all time averages over 5 iterations)                        |                         |
 +---------------------------------------------------------------------------+-------------------------+
 | The total execution time of all queries is:                               | 472.02 ms               |
 |   (The sum of all time averages over 5 iterations)                        |                         |
 +---------------------------------------------------------------------------+-------------------------+


 # An overview of the queries that finished with their execution time:
 +-----------------+------+---------+---------------+----------------+
 |   QUERY NAME    | Done | Runtime | Planning Time | Execution Time |
 +-----------------+------+---------+---------------+----------------+
 | test_1          |  x   | -       | 0.029         | 0.014          |
 +-----------------+------+---------+---------------+----------------+
 | insight_1       |  x   | -       | 0.021         | 0.66           |
 +-----------------+------+---------+---------------+----------------+
 | insight_2       |  x   | -       | 0.03          | 0.801          |
 +-----------------+------+---------+---------------+----------------+
 | insight_3       |  x   | -       | 0.046         | 1.583          |
 +-----------------+------+---------+---------------+----------------+
 | insight_4       |  x   | -       | 0.056         | 0.775          |
 +-----------------+------+---------+---------------+----------------+
 | insight_5       |  x   | -       | 0.044         | 0.542          |
 +-----------------+------+---------+---------------+----------------+
 | insight_6       |  x   | -       | 0.096         | 102.66         |
 +-----------------+------+---------+---------------+----------------+
 | probabilistic_1 |  x   | -       | 0.097         | 102.78         |
 +-----------------+------+---------+---------------+----------------+
 | probabilistic_2 |  x   | -       | 0.079         | 101.2          |
 +-----------------+------+---------+---------------+----------------+
 | probabilistic_3 |  x   | -       | 0.09          | 103.02         |
 +-----------------+------+---------+---------------+----------------+
 | probabilistic_4 |      | -       | -             | -              |
 +-----------------+------+---------+---------------+----------------+
 | probabilistic_5 |  x   | -       | 0.47          | 46.812         |
 +-----------------+------+---------+---------------+----------------+
 | probabilistic_6 |  x   | -       | 0.579         | 11.178         |
 +-----------------+------+---------+---------------+----------------+



 # ==== Overview of errors and possible missing functionality. ==== #
 # Based on the errors thrown during benchmark testing, the batabase system might
 # lack support for one or more functionalities. Please also verify the actual error
 # thrown by the database manually, as that may provide a clearer indication of what
 # went wrong during benchmarking. If a memory allocation error is thrown, you can
 # alter the query to run it on the 'part' table to see if the functionality of the
 # query is supported.

 # The following queries have raised an error:

 QUERY #:                probabilistic_4
 Functionality message:  The DBMS might lack the ability to calculate the probability of a composed query result.
                        It may struggle composing large and/or complex sentences/probability spaces.
 Error raised by DBMS:
column "timeout" does not exist
LINE 3:         SELECT timeout, category, AGG_OR(_sentence) AS sente...
                       ^
```

```
     # ============================================= #
     # ==============   QuestionMark  ============== #
     # ============================================= #
     # The metrics file.
     # Run on DuBio.

This file contains the metrics of this benchmark test.
Please see 'QM_query_results.txt' for the results and runtimes of the queries.

The included metrics produced the following results:
+---------------------------------------------------------------------------+-------------------------+
|        METRIC                                                             |       VALUE             |
+---------------------------------------------------------------------------+-------------------------+
| The total size of the _sentence column is:                                | 160 kB                  |
+---------------------------------------------------------------------------+-------------------------+
| The total size of the dict table is:                                      | 784 kB                  |
+---------------------------------------------------------------------------+-------------------------+
| The percentage of data used for probabilistic representation:             | 50.86%                  |
+---------------------------------------------------------------------------+-------------------------+
| The total number of characters needed for all queries:                    | 2138 characters         |
+---------------------------------------------------------------------------+-------------------------+
| The percentage of successful queries is:                                  | 100.0%                  |
|   (See below what functionality might be lacking)                         |                         |
+---------------------------------------------------------------------------+-------------------------+
| The total planning time of all queries is:                                | 0.26 ms                 |
|   (The sum of all time averages over 5 iterations)                        |                         |
+---------------------------------------------------------------------------+-------------------------+
| The total execution time of all queries is:                               | 10.63 ms                |
|   (The sum of all time averages over 5 iterations)                        |                         |
+---------------------------------------------------------------------------+-------------------------+


# An overview of the queries that finished with their execution time:
+-----------------+------+---------+---------------+----------------+
|   QUERY NAME    | Done | Runtime | Planning Time | Execution Time |
+-----------------+------+---------+---------------+----------------+
| IUD_1_rollback  |  x   | -       | 0.067         | 2.298          |
+-----------------+------+---------+---------------+----------------+
| IUD_2_rollback  |  x   | -       | 0.069         | 4.908          |
+-----------------+------+---------+---------------+----------------+
| IUD_3_rollback  |  x   | -       | 0.041         | 1.128          |
+-----------------+------+---------+---------------+----------------+
| IUD_4_rollback  |  x   | -       | 0.043         | 1.104          |
+-----------------+------+---------+---------------+----------------+
| IUD_5_rollback  |  x   | -       | 0.037         | 1.194          |
+-----------------+------+---------+---------------+----------------+
      # =========================================== #
      # ==============   QuestionMark  ============== #
      # =========================================== #
      # The query results file.
      # Run on MayBMS.

This file contains the query results and runtimes of this benchmark test.
The query plan and average run time are produced by PostgreSQL EXPLAIN ANALYSE.
Please see 'QM_metrics_results' for the results of the metrics.


# ============== test_1 ============== #

    SELECT id
    FROM offers
    LIMIT 10;

+-------+--------+---------+-----+--------+---------+------+
| id    | _v0    | _d0     | _p0 | _v1    | _d1     | _p1  |
+-------+--------+---------+-----+--------+---------+------+
| 16993 | 393573 | 1092388 | 1.0 | 392570 | 1090230 | 0.2  |
| 20537 | 393574 | 1092808 | 1.0 | 392836 | 1090650 | 0.5  |
| 27050 | 393575 | 1092606 | 1.0 | 392835 | 1090448 | 0.5  |
| 38200 | 393576 | 1093463 | 1.0 | 392699 | 1091305 | 1.0  |
| 59437 | 393577 | 1092522 | 1.0 | 392940 | 1090364 | 1.0  |
| 71688 | 393578 | 1092798 | 1.0 | 392480 | 1090640 | 1.0  |
| 86426 | 393579 | 1093049 | 1.0 | 393025 | 1090891 | 0.25 |
| 87222 | 393580 | 1093366 | 1.0 | 392869 | 1091208 | 0.2  |
| 88301 | 393581 | 1092456 | 1.0 | 393515 | 1090298 | 0.2  |
| 88391 | 393582 | 1091682 | 1.0 | 392987 | 1089524 | 0.5  |
+-------+--------+---------+-----+--------+---------+------+

Average total time over 5 iterations: 0.031 ms.
+-------------------------------------------------------------------------------------+
| QUERY PLAN                                                                          |
+-------------------------------------------------------------------------------------+
| Limit  (cost=0.00..0.79 rows=10 width=32) (actual time=0.005..0.028 rows=10 loops=1) |
+-------------------------------------------------------------------------------------+




# ============== insight_1 ============== #

    SELECT *
    FROM offers;

+----+------------+-------+-------+----------+-------------+-------+--------------+-----+
| id | cluster_id | title | brand | category | description | price | identifiers  | ... |
+----+------------+-------+-------+----------+-------------+-------+--------------+-----+
|    |            |       |       |          |             |       |              |     |
+----+------------+-------+-------+----------+-------------+-------+--------------+-----+
The first 20 out of 7947 rows are shown.
Some returned records were too large to display. This query returned 7947 rows.

Average total time over 5 iterations: 1.521 ms.
+--------------------------------+
| This query returned no records. |
+--------------------------------+




# ============== insight_2 ============== #

    SELECT COUNT(*) as records,
        COUNT(DISTINCT(id)) as offers,
        COUNT(DISTINCT(cluster_id)) as clusters
    FROM offers_setup;

+---------+--------+----------+
| records | offers | clusters |
+---------+--------+----------+
| 2311    | 1653   | 1229     |
+---------+--------+----------+

Average total time over 5 iterations: 2.272 ms.
+-------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                |
+-------------------------------------------------------------------------------------------+
| Aggregate  (cost=185.45..185.46 rows=1 width=16) (actual time=2.738..2.738 rows=1 loops=1) |
+-------------------------------------------------------------------------------------------+
# ============== insight_3 ============== #

    SELECT cluster_size, COUNT(cluster_size) as amount
    FROM (
        SELECT COUNT(DISTINCT(id)) as cluster_size
        FROM offers_setup
        GROUP BY cluster_id
    ) as cluster_sizes
    GROUP BY cluster_size
    ORDER BY cluster_size ASC;

+--------------+--------+
| cluster_size | amount |
+--------------+--------+
| 1            | 886    |
| 2            | 187    |
| 3            | 72     |
| 4            | 41     |
| 5            | 43     |
+--------------+--------+

Average total time over 5 iterations: 4.039 ms.
+------------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                       |
+------------------------------------------------------------------------------------------------------------------+
| Sort  (cost=358.50..359.00 rows=200 width=8) (actual time=3.996..3.997 rows=5 loops=1)                           |
|   Sort Key: (count(DISTINCT offers_setup.id))                                                                    |
|   Sort Method:  quicksort  Memory: 25kB                                                                          |
|   ->  HashAggregate  (cost=348.36..350.86 rows=200 width=8) (actual time=3.990..3.993 rows=5 loops=1)            |
|         ->  GroupAggregate  (cost=297.23..329.92 rows=1229 width=16) (actual time=1.274..3.690 rows=1229 loops=1)|
|               ->  Sort  (cost=297.23..303.01 rows=2311 width=16) (actual time=1.267..1.403 rows=2311 loops=1)    |
|                     Sort Key: offers_setup.cluster_id                                                            |
|                     Sort Method:  quicksort  Memory: 205kB                                                       |
+------------------------------------------------------------------------------------------------------------------+
```

```
# ============== insight_4 ============== #

    SELECT ROUND(all_certain::decimal / all_offers::decimal, 4) * 100 AS certain_percentage
    FROM (
        SELECT COUNT(id) AS all_offers
        FROM offers_setup
    ) AS count_all, (
        SELECT COUNT(id) AS all_certain
        FROM (
            SELECT id, tconf() AS confidence
            FROM offers
        ) AS confidences
        WHERE confidence = 1
    ) AS count_cert;

+--------------------+
| certain_percentage |
+--------------------+
| 32.5400            |
+--------------------+

Average total time over 5 iterations: 9.817 ms.
+------------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                       |
+------------------------------------------------------------------------------------------------------------------+
| Nested Loop  (cost=881.93..882.00 rows=1 width=16) (actual time=9.477..9.478 rows=1 loops=1)                     |
|   ->  Aggregate  (cost=173.89..173.90 rows=1 width=8) (actual time=0.652..0.652 rows=1 loops=1)                  |
|         ->  Seq Scan on offers_setup  (cost=0.00..168.11 rows=2311 width=8) (actual time=0.006..0.379 rows=2311  |
|                                                                                                         loops=1) |
|   ->  Aggregate  (cost=708.04..708.05 rows=1 width=8) (actual time=8.816..8.816 rows=1 loops=1)                  |
|         ->  Seq Scan on offers  (cost=0.00..707.94 rows=40 width=8) (actual time=0.009..8.734 rows=752 loops=1)  |
+------------------------------------------------------------------------------------------------------------------+
```

```
# ============== insight_5 ============== #

    SELECT id, tconf(*), _v0
    FROM offers
    WHERE _v1 = 52379
    AND _d1 = 548185;

+---------------------------------+
| This query returned no records. |
+---------------------------------+
Average total time over 5 iterations: 3.624 ms.
+---------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                        |
+---------------------------------------------------------------------------------------------------+
| Seq Scan on offers  (cost=0.00..668.21 rows=1 width=32) (actual time=2.582..2.582 rows=0 loops=1) |
+---------------------------------------------------------------------------------------------------+


# ============== insight_6 ============== #

    SELECT round((AVG(tconf()) * 100)::NUMERIC, 4) AS certainty_of_the_dataset
    FROM offers;

+--------------------------+
| certainty_of_the_dataset |
+--------------------------+
| 17.3665                  |
+--------------------------+

Average total time over 5 iterations: 9.11 ms.
+--------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                 |
+--------------------------------------------------------------------------------------------+
| Aggregate  (cost=648.34..648.37 rows=1 width=24) (actual time=9.141..9.141 rows=1 loops=1) |
+--------------------------------------------------------------------------------------------+


# ============== probabilistic_1 ============== #

    SELECT round(tconf()::decimal, 4) AS probability, *
    FROM offers
    ORDER BY probability DESC;


+-------------+----+------------+-------+-------+----------+-------------+-------+--------------+-----+
| probability | id | cluster_id | title | brand | category | description | price | identifiers  | ... |
+-------------+----+------------+-------+-------+----------+-------------+-------+--------------+-----+
|             |    |            |       |       |          |             |       |              |     |
+-------------+----+------------+-------+-------+----------+-------------+-------+--------------+-----+
The first 20 out of 7947 rows are shown.
Some returned records were too large to display. This query returned 7947 rows.

Average total time over 5 iterations: 49.65 ms.
+--------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                       |
+--------------------------------------------------------------------------------------------------+
| Sort  (cost=3062.62..3082.49 rows=7947 width=511) (actual time=38.528..43.989 rows=7947 loops=1) |
|   Sort Key: (round((tconf((tconf(_v0, _d0, _p0, _v1, _d1, _p1) * 1::real)))::numeric, 4))        |
|   Sort Method:  external merge  Disk: 3920kB                                                     |
+--------------------------------------------------------------------------------------------------+


# ============== probabilistic_2 ============== #

    SELECT category, ECOUNT() AS expected_count
    FROM offers
    GROUP BY category
    ORDER BY expected_count DESC;

+----------------------------+----------------+
| category                   | expected_count |
+----------------------------+----------------+
| Tools_and_Home_Improvement | 161.539        |
| Home_and_Garden            | 137.941        |
| Office_Products            | 133.837        |
| Clothing                   | 118.032        |
| Automotive                 | 104.643        |
| Other_Electronics          | 76.658         |
| Books                      | 62.8282        |
| Shoes                      | 61.7045        |
| Sports_and_Outdoors        | 56.8202        |
| Computers_and_Accessories  | 51.571         |
| Musical_Instruments        | 48.5879        |
| Jewelry                    | 45.7761        |
| Health_and_Beauty          | 42.7384        |
| Grocery_and_Gourmet_Food   | 37.38          |
| Camera_and_Photo           | 35.6586        |
| Luggage_and_Travel_Gear    | 34.4583        |
| Toys_and_Games             | 31.8327        |
| CDs_and_Vinyl              | 28.6636        |
| Cellphones_and_Accessories | 25.75          |
+----------------------------+----------------+
The first 20 out of 25 rows are shown.
```

```
Average total time over 5 iterations: 11.338 ms.
+----------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                               |
+----------------------------------------------------------------------------------------------------------+
| Sort  (cost=669.29..669.35 rows=25 width=39) (actual time=10.914..10.916 rows=25 loops=1)                |
|   Sort Key: (sum(tconf((tconf(_v0, _d0, _p0, _v1, _d1, _p1) * 1::real))))                                |
|   Sort Method:  quicksort  Memory: 26kB                                                                  |
|   ->  HashAggregate  (cost=668.20..668.70 rows=25 width=39) (actual time=10.893..10.900 rows=25 loops=1) |
+----------------------------------------------------------------------------------------------------------+




# ============== probabilistic_3 ============== #

    SELECT cluster_id, esum(id), COUNT(id) AS number_of_offers
    FROM offers
    GROUP BY cluster_id
    ORDER BY number_of_offers DESC;

+------------+-------------+------------------+
| cluster_id | esum        | number_of_offers |
+------------+-------------+------------------+
| 81         | 38858800.0  | 780              |
| 30         | 88758700.0  | 705              |
| 41         | 40992700.0  | 616              |
| 57         | 39380500.0  | 352              |
| 177        | 18318000.0  | 234              |
| 31         | 111037000.0 | 210              |
| 42         | 70598000.0  | 182              |
| 82         | 45020700.0  | 180              |
| 49         | 58874300.0  | 176              |
| 32         | 47036100.0  | 150              |
| 140        | 19146800.0  | 140              |
| 46         | 15675100.0  | 140              |
| 17         | 26618500.0  | 133              |
| 162        | 13456800.0  | 133              |
| 58         | 114070000.0 | 125              |
| 43         | 53461800.0  | 112              |
| 83         | 9304770.0   | 105              |
| 59         | 106551000.0 | 94               |
| 178        | 23974600.0  | 84               |
+------------+-------------+------------------+
The first 20 out of 1229 rows are shown.

Average total time over 5 iterations: 14.41 ms.
+-------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                  |
+-------------------------------------------------------------------------------------------------------------+
| Sort  (cost=744.35..746.17 rows=726 width=40) (actual time=15.869..15.935 rows=1229 loops=1)                |
|   Sort Key: (count(id))                                                                                     |
|   Sort Method:  quicksort  Memory: 145kB                                                                    |
|   ->  HashAggregate  (cost=688.07..709.85 rows=726 width=40) (actual time=15.128..15.547 rows=1229 loops=1) |
+-------------------------------------------------------------------------------------------------------------+




# ============== probabilistic_4 ============== #

    SELECT category, conf() AS probability, variables
    FROM offers
    GROUP BY category
    ORDER BY probability DESC;


The following error occurred while executing this query:
column "variables" does not exist
LINE 2:     SELECT category, conf() AS probability, variables
                                                    ^




# ============== probabilistic_5 ============== #

    SELECT *, round(tconf()::NUMERIC, 4) AS probability
    FROM offers
    WHERE cluster_id IN (
        SELECT cluster_id
        FROM offers_setup
        WHERE title LIKE '%card%'
        OR description LIKE '%card%'
    )
    ORDER BY probability DESC
    LIMIT 1;
```

```
+----+------------+-------+-------+----------+-------------+-------+-------------+-----+
| id | cluster_id | title | brand | category | description | price | identifiers | ... |
+----+------------+-------+-------+----------+-------------+-------+-------------+-----+
|    |            |       |       |          |             |       |             |     |
+----+------------+-------+-------+----------+-------------+-------+-------------+-----+
Some returned records were too large to display. This query returned 1 rows.

Average total time over 5 iterations: 13.316 ms.
+-----------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                |
+-----------------------------------------------------------------------------------------------------------+
| Limit  (cost=870.09..870.09 rows=1 width=511) (actual time=13.137..13.137 rows=1 loops=1)                 |
|   ->  Sort  (cost=870.09..871.27 rows=472 width=511) (actual time=13.137..13.137 rows=1 loops=1)          |
|         Sort Key: (round((tconf((tconf(offers._v0, offers._d0, offers._p0, offers._v1, offers._d1, offers._p1) * |
|                                                                                1::real)))::numeric, 4)) |
|         Sort Method:  top-N heapsort  Memory: 26kB                                                        |
|         ->  Hash IN Join  (cost=181.38..867.73 rows=472 width=511) (actual time=2.917..11.272 rows=2880 loops=1) |
|               Hash Cond: (offers.cluster_id = offers_setup.cluster_id)                                    |
|               ->  Seq Scan on offers  (cost=0.00..628.47 rows=7947 width=511) (actual time=0.002..1.125 rows=7947|
|                                                                                                loops=1) |
|               ->  Hash  (cost=179.66..179.66 rows=137 width=8) (actual time=2.896..2.896 rows=153 loops=1)  |
|                     ->  Seq Scan on offers_setup  (cost=0.00..179.66 rows=137 width=8) (actual time=0.102..2.859 |
|                                                                                       rows=153 loops=1) |
+-----------------------------------------------------------------------------------------------------------+
```

```
# ============= probabilistic_6 ============= #

    SELECT id, cluster_id, brand, category, identifiers
    FROM offers
    WHERE title LIKE '%card%'
    OR description LIKE '%card%'
    AND tconf() > 0.45
    AND tconf() < 0.55;

+---------+------------+-----+--------+---------+-----------+--------+---------+------------+
| id      | cluster_id | ... | _v0    | _d0     | _p0       | _v1    | _d1     | _p1        |
+---------+------------+-----+--------+---------+-----------+--------+---------+------------+
| 1143303 | 1182       |     | 393672 | 1092102 | 1.0       | 392371 | 1089944 | 0.2        |
| 2148295 | 1032       |     | 393770 | 1092196 | 1.0       | 393399 | 1090038 | 0.2        |
| 2548259 | 394        |     | 393801 | 1093712 | 1.0       | 393107 | 1091554 | 1.0        |
| 2650851 | 304        |     | 393810 | 1091695 | 1.0       | 392842 | 1089537 | 1.0        |
| 3007245 | 30         |     | 393839 | 1092863 | 0.054431  | 393005 | 1090705 | 0.00710873 |
| 3007245 | 30         |     | 393839 | 1092863 | 0.054431  | 393005 | 1089648 | 0.0144388  |
| 3007245 | 30         |     | 393839 | 1092863 | 0.054431  | 393005 | 1090951 | 0.0165973  |
| 3007245 | 30         |     | 393839 | 1092863 | 0.054431  | 393005 | 1090951 | 0.0165973  |
| 3007245 | 30         |     | 393839 | 1092863 | 0.054431  | 393005 | 1091526 | 0.0336988  |
| 3007245 | 30         |     | 393839 | 1092863 | 0.054431  | 393005 | 1091526 | 0.0336988  |
| 3007245 | 30         |     | 393839 | 1092863 | 0.054431  | 393005 | 1091605 | 0.038729   |
| 3007245 | 30         |     | 393839 | 1092863 | 0.054431  | 393005 | 1089958 | 0.0786326  |
| 3007245 | 30         |     | 393839 | 1092863 | 0.054431  | 392378 | 1091570 | 0.25       |
| 3007245 | 30         |     | 393839 | 1092863 | 0.054431  | 392378 | 1091570 | 0.25       |
| 3007245 | 30         |     | 393839 | 1092863 | 0.054431  | 392378 | 1091570 | 0.25       |
| 3007245 | 30         |     | 393839 | 1092863 | 0.054431  | 392378 | 1091570 | 0.25       |
| 3007245 | 30         |     | 393839 | 1092863 | 0.054431  | 392418 | 1089508 | 0.333333   |
| 3007245 | 30         |     | 393839 | 1092863 | 0.054431  | 392418 | 1089508 | 0.333333   |
| 3007245 | 30         |     | 393839 | 1092863 | 0.054431  | 392418 | 1089508 | 0.333333   |
+---------+------------+-----+--------+---------+-----------+--------+---------+------------+
The first 20 out of 838 rows are shown.
Some returned records were too large to display. This query returned 838 rows.

Average total time over 5 iterations: 3.637 ms.
+-------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                      |
+-------------------------------------------------------------------------------------------------+
| Seq Scan on offers  (cost=0.00..648.34 rows=707 width=101) (actual time=0.355..3.325 rows=838 loops=1) |
+-------------------------------------------------------------------------------------------------+
      # ============================================= #
      # ==============   QuestionMark  ============== #
      # ============================================= #
      # The query results file.
      # Run on MayBMS.

This file contains the query results and runtimes of this benchmark test.
The query plan and average run time are produced by PostgreSQL EXPLAIN ANALYSE.
Please see 'QM_metrics_results' for the results of the metrics.


# ============== IUD_1_rollback ============== #

    INSERT INTO offers_setup (id, cluster_id, title, brand, category, description, price, identifiers,
                              keyvaluepairs, spectablecontent, world_prob, attribute_prob)
        VALUES(-464, 77, ..., 0.42911, 0.629),
              (-466, 77, ..., 0.5, 0.246),
              (-468, 77, ..., 0.32635, 0.629),
              (-469, 77, ..., 0.5, 0.125),
              (-471, 77, ..., 0.24454, 0.629);

Average total time over 5 iterations: 0.569 ms.
+---------------------------------+
| This query returned no records. |
+---------------------------------+




# ============== IUD_2_rollback ============== #

    INSERT INTO offers_setup (id, cluster_id, title, brand, category, description, price, identifiers,
                              keyvaluepairs, spectablecontent, world_prob, attribute_prob)
    SELECT * FROM bulk_insert;

Average total time over 5 iterations: 106.54 ms.
+---------------------------------+
| This query returned no records. |
+---------------------------------+




# ============== IUD_3_rollback ============== #

    UPDATE offers
    SET world_prob = 0.345,
        attribute_prob = 0.3992
    WHERE _d0 = 615777
    AND _d1 = 613619;

    UPDATE offers
    SET world_prob = 0.345,
        attribute_prob = 0.6008
    WHERE _d0 = 615777
    and _d1 = 613841;

    UPDATE offers
    SET world_prob = 0.1254,
        attribute_prob = 0.5
    WHERE _d0 = 615999
    AND _d1 = 613619;

    UPDATE offers
    SET world_prob = 0.1254,
        attribute_prob = 0.5
    WHERE _d0 = 615999
    and _d1 = 613841;

    UPDATE offers
    SET world_prob = 0.487,
        attribute_prob = 0.4300
    WHERE _d0 = 615850
    and _d1 = 613395;

    UPDATE offers
    SET world_prob = 0.487,
        attribute_prob = 0.5700
    WHERE _d0 = 615850
    AND _d1 = 613692;
    UPDATE offers
    SET world_prob = 0.487,
        attribute_prob = 0.4300
    WHERE _d0 = 615999
    and _d1 = 613841;

    UPDATE offers
    SET world_prob = 0.487
    WHERE _d0 = 615553
    AND _d1 = 613692;

    UPDATE offers
    SET world_prob = 0.487
    WHERE _d0 = 615553
    and _d1 = 613395;

    UPDATE offers
    SET world_prob = 0.329
    WHERE _d0 = 614032
    AND _d1 = 612733;

    UPDATE offers
    SET world_prob = 0.329
    WHERE _d0 = 614032
    and _d1 = 611874;

    UPDATE offers
    SET world_prob = 0.184
    WHERE _d0 = 615197
    AND _d1 = 612733;

    UPDATE offers
    SET world_prob = 0.184
    WHERE _d0 = 615197
    and _d1 = 613039;
Average total time over 5 iterations: 2.508 ms.
+----------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                         |
+----------------------------------------------------------------------------------------------------+
| Seq Scan on offers  (cost=0.00..668.21 rows=1 width=501) (actual time=2.501..2.501 rows=0 loops=1) |
+----------------------------------------------------------------------------------------------------+

Average total time over 5 iterations: 2.494 ms.
+----------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                         |
+----------------------------------------------------------------------------------------------------+
| Seq Scan on offers  (cost=0.00..668.21 rows=1 width=501) (actual time=2.533..2.533 rows=0 loops=1) |
+----------------------------------------------------------------------------------------------------+

Average total time over 5 iterations: 2.576 ms.
+----------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                         |
+----------------------------------------------------------------------------------------------------+
| Seq Scan on offers  (cost=0.00..668.21 rows=1 width=501) (actual time=2.453..2.453 rows=0 loops=1) |
+----------------------------------------------------------------------------------------------------+

Average total time over 5 iterations: 2.577 ms.
+----------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                         |
+----------------------------------------------------------------------------------------------------+
| Seq Scan on offers  (cost=0.00..668.21 rows=1 width=501) (actual time=2.542..2.542 rows=0 loops=1) |
+----------------------------------------------------------------------------------------------------+

Average total time over 5 iterations: 2.708 ms.
+----------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                         |
+----------------------------------------------------------------------------------------------------+
| Seq Scan on offers  (cost=0.00..668.21 rows=1 width=501) (actual time=2.750..2.750 rows=0 loops=1) |
+----------------------------------------------------------------------------------------------------+

Average total time over 5 iterations: 2.852 ms.
+----------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                         |
+----------------------------------------------------------------------------------------------------+
| Seq Scan on offers  (cost=0.00..668.21 rows=1 width=501) (actual time=2.974..2.974 rows=0 loops=1) |
+----------------------------------------------------------------------------------------------------+

Average total time over 5 iterations: 2.925 ms.
+----------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                         |
+----------------------------------------------------------------------------------------------------+
| Seq Scan on offers  (cost=0.00..668.21 rows=1 width=501) (actual time=3.075..3.075 rows=0 loops=1) |
+----------------------------------------------------------------------------------------------------+
Average total time over 5 iterations: 2.995 ms.
+----------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                         |
+----------------------------------------------------------------------------------------------------+
| Seq Scan on offers  (cost=0.00..668.21 rows=1 width=509) (actual time=2.988..2.988 rows=0 loops=1) |
+----------------------------------------------------------------------------------------------------+

Average total time over 5 iterations: 2.849 ms.
+----------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                         |
+----------------------------------------------------------------------------------------------------+
| Seq Scan on offers  (cost=0.00..668.21 rows=1 width=509) (actual time=2.990..2.990 rows=0 loops=1) |
+----------------------------------------------------------------------------------------------------+

Average total time over 5 iterations: 2.765 ms.
+----------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                         |
+----------------------------------------------------------------------------------------------------+
| Seq Scan on offers  (cost=0.00..668.21 rows=1 width=509) (actual time=2.781..2.781 rows=0 loops=1) |
+----------------------------------------------------------------------------------------------------+

Average total time over 5 iterations: 2.813 ms.
+----------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                         |
+----------------------------------------------------------------------------------------------------+
| Seq Scan on offers  (cost=0.00..668.21 rows=1 width=509) (actual time=2.636..2.636 rows=0 loops=1) |
+----------------------------------------------------------------------------------------------------+

Average total time over 5 iterations: 2.593 ms.
+----------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                         |
+----------------------------------------------------------------------------------------------------+
| Seq Scan on offers  (cost=0.00..668.21 rows=1 width=509) (actual time=2.626..2.626 rows=0 loops=1) |
+----------------------------------------------------------------------------------------------------+

Average total time over 5 iterations: 2.56 ms.
+----------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                         |
+----------------------------------------------------------------------------------------------------+
| Seq Scan on offers  (cost=0.00..668.21 rows=1 width=509) (actual time=2.469..2.469 rows=0 loops=1) |
+----------------------------------------------------------------------------------------------------+



# ============= IUD_4_rollback ============= #

    UPDATE offers_setup
    SET cluster_id = max_cluster.max_id,
        world_prob = 1,
        attribute_prob = 1
    FROM (
        SELECT max(cluster_id) + 1 AS max_id
        FROM offers_setup
    ) as max_cluster
    WHERE id = 12071001;

    UPDATE offers_setup
    SET cluster_id = max_cluster.max_id,
        world_prob = 1,
        attribute_prob = 1
    FROM (
        SELECT max(cluster_id) + 1 AS max_id
        FROM offers_setup
    ) as max_cluster
    WHERE id = 16457529;

    UPDATE offers_setup
    SET world_prob = 0.63,
        attribute_prob = 0.5
    WHERE id = 7339350;

    UPDATE offers_setup
    SET world_prob = 0.63,
        attribute_prob = 0.5
    WHERE id = 12326926;

Average total time over 5 iterations: 1.692 ms.
+----------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                     |
+----------------------------------------------------------------------------------------------------------------+
| Nested Loop  (cost=885.03..1770.09 rows=1 width=440) (actual time=1.325..1.637 rows=1 loops=1)                 |
|   ->  Seq Scan on offers_setup  (cost=0.00..885.02 rows=1 width=432) (actual time=0.328..0.640 rows=1 loops=1) |
|         Filter: (id = 12071001)                                                                                |
|   ->  Aggregate  (cost=885.03..885.04 rows=1 width=8) (actual time=0.993..0.993 rows=1 loops=1)                |
+----------------------------------------------------------------------------------------------------------------+
Average total time over 5 iterations: 1.724 ms.
+----------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                     |
+----------------------------------------------------------------------------------------------------------------+
| Nested Loop  (cost=885.03..1770.09 rows=1 width=440) (actual time=1.480..1.810 rows=1 loops=1)                 |
|   ->  Seq Scan on offers_setup  (cost=0.00..885.02 rows=1 width=432) (actual time=0.359..0.689 rows=1 loops=1) |
|         Filter: (id = 16457529)                                                                                |
|   ->  Aggregate  (cost=885.03..885.04 rows=1 width=8) (actual time=1.118..1.118 rows=1 loops=1)                |
+----------------------------------------------------------------------------------------------------------------+

Average total time over 5 iterations: 0.746 ms.
+----------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                               |
+----------------------------------------------------------------------------------------------------------+
| Seq Scan on offers_setup  (cost=0.00..886.23 rows=1 width=440) (actual time=0.346..0.768 rows=1 loops=1) |
+----------------------------------------------------------------------------------------------------------+

Average total time over 5 iterations: 0.68 ms.
+----------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                               |
+----------------------------------------------------------------------------------------------------------+
| Seq Scan on offers_setup  (cost=0.00..887.42 rows=1 width=440) (actual time=0.330..0.740 rows=1 loops=1) |
+----------------------------------------------------------------------------------------------------------+



# ============== IUD_5_rollback ============== #

    DELETE FROM offers_setup
    WHERE cluster_id = 41;

Average total time over 5 iterations: 0.703 ms.
+-----------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                |
+-----------------------------------------------------------------------------------------------------------+
| Seq Scan on offers_setup  (cost=0.00..887.42 rows=225 width=6) (actual time=0.068..0.838 rows=44 loops=1) |
+-----------------------------------------------------------------------------------------------------------+
      # ============================================== #
      # ==============   QuestionMark  ============== #
      # ============================================== #
      # The query results file.
      # Run on DuBio.

This file contains the query results and runtimes of this benchmark test.
The query plan and average run time are produced by PostgreSQL EXPLAIN ANALYSE.
Please see 'QM_metrics_results' for the results of the metrics.


# ============== test_1 ============== #

    SELECT id
    FROM offers
    LIMIT 10;

+----------+
| id       |
+----------+
| 12270851 |
| 15769078 |
| 10571389 |
| 9389315  |
| 14030659 |
| 11592947 |
| 15165064 |
| 3304586  |
| 11698262 |
| 15756894 |
+----------+


Average planning time over 5 iterations:  0.029 ms.
Average execution time over 5 iterations: 0.014 ms.

+------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                 |
+------------------------------------------------------------------------------------------------------------+
| Limit  (cost=0.00..0.78 rows=10 width=8) (actual time=0.002..0.004 rows=10 loops=1)                        |
|   ->  Seq Scan on offers  (cost=0.00..135.25 rows=1725 width=8) (actual time=0.002..0.003 rows=10 loops=1) |
+------------------------------------------------------------------------------------------------------------+




# ============== insight_1 ============== #

    SELECT *
    FROM offers;

    SELECT print(dict) FROM _dict WHERE name='mydict';

+-----------------------------------------------------+
| print                                               |
+-----------------------------------------------------+
+-----------------------------------------------------+
Some returned records were too large to display. This query returned 1 rows.


Average planning time over 5 iterations:  0.035 ms.
Average execution time over 5 iterations: 0.288 ms.

+----------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                               |
+----------------------------------------------------------------------------------------------------------+
| Seq Scan on offers  (cost=0.00..135.25 rows=1725 width=511) (actual time=0.002..0.106 rows=1725 loops=1) |
+----------------------------------------------------------------------------------------------------------+


Average planning time over 5 iterations:  0.021 ms.
Average execution time over 5 iterations: 0.66 ms.

+-------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                      |
+-------------------------------------------------------------------------------------------------+
| Seq Scan on _dict  (cost=0.00..33.02 rows=1 width=32) (actual time=0.963..0.965 rows=1 loops=1) |
|   Filter: ((name)::text = 'mydict'::text)                                                       |
+-------------------------------------------------------------------------------------------------+
# ============== insight_2 ============== #

    SELECT COUNT(*) as records,
        COUNT(DISTINCT(id)) as offers,
        COUNT(DISTINCT(cluster_id)) as clusters
    FROM offers;

+---------+--------+----------+
| records | offers | clusters |
+---------+--------+----------+
| 1725    | 1653   | 1111     |
+---------+--------+----------+


Average planning time over 5 iterations:  0.03 ms.
Average execution time over 5 iterations: 0.801 ms.

+--------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                   |
+--------------------------------------------------------------------------------------------------------------+
| Aggregate  (cost=148.19..148.20 rows=1 width=24) (actual time=0.485..0.486 rows=1 loops=1)                   |
|   ->  Seq Scan on offers  (cost=0.00..135.25 rows=1725 width=16) (actual time=0.002..0.110 rows=1725 loops=1) |
+--------------------------------------------------------------------------------------------------------------+
```

```
# ============== insight_3 ============== #

    SELECT cluster_size, COUNT(cluster_size) as amount
    FROM (
        SELECT COUNT(DISTINCT(id)) as cluster_size
        FROM offers
        GROUP BY cluster_id
    ) as cluster_sizes
    GROUP BY cluster_size
    ORDER BY cluster_size ASC;

+--------------+--------+
| cluster_size | amount |
+--------------+--------+
| 1            | 824    |
| 2            | 126    |
| 3            | 56     |
| 4            | 44     |
| 5            | 61     |
+--------------+--------+


Average planning time over 5 iterations:  0.046 ms.
Average execution time over 5 iterations: 1.583 ms.

+------------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                       |
+------------------------------------------------------------------------------------------------------------------+
| Sort  (cost=278.35..278.85 rows=200 width=16) (actual time=1.262..1.263 rows=5 loops=1)                          |
|   Sort Key: (count(DISTINCT offers.id))                                                                          |
|   Sort Method: quicksort  Memory: 25kB                                                                           |
|   ->  HashAggregate  (cost=268.70..270.70 rows=200 width=16) (actual time=1.259..1.260 rows=5 loops=1)           |
|         Group Key: count(DISTINCT offers.id)                                                                     |
|         Batches: 1  Memory Usage: 40kB                                                                           |
|         ->  GroupAggregate  (cost=227.99..252.04 rows=1111 width=16) (actual time=0.355..1.121 rows=1111 loops=1)|
|               Group Key: offers.cluster_id                                                                       |
|               ->  Sort  (cost=227.99..232.30 rows=1725 width=16) (actual time=0.349..0.406 rows=1725 loops=1)    |
|                     Sort Key: offers.cluster_id                                                                  |
|                     Sort Method: quicksort  Memory: 143kB                                                        |
|                     ->  Seq Scan on offers  (cost=0.00..135.25 rows=1725 width=16) (actual time=0.003..0.174     |
|                                                                                               rows=1725 loops=1) |
+------------------------------------------------------------------------------------------------------------------+
```

```
# ============== insight_4 ============== #

    SELECT ROUND(COUNT(CASE WHEN istrue(_sentence) THEN 1 END)::decimal / COUNT(*)::decimal, 4) * 100 AS
certain_percentage
    FROM offers;

+--------------------+
| certain_percentage |
+--------------------+
| 47.7700            |
+--------------------+
Average planning time over 5 iterations:  0.056 ms.
Average execution time over 5 iterations: 0.775 ms.

+--------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                   |
+--------------------------------------------------------------------------------------------------------------+
| Aggregate  (cost=148.19..148.21 rows=1 width=32) (actual time=0.368..0.368 rows=1 loops=1)                   |
|   ->  Seq Scan on offers  (cost=0.00..135.25 rows=1725 width=94) (actual time=0.002..0.094 rows=1725 loops=1) |
+--------------------------------------------------------------------------------------------------------------+




# ============== insight_5 ============== #

    SELECT o.id, prob(dict, 'w43=1') AS probability, hasrva(_sentence, 'w43=1')
    FROM offers o, _dict d
    WHERE hasrva(_sentence, 'w43=1');

+----------+--------------------+--------+
| id       | probability        | hasrva |
+----------+--------------------+--------+
| 11119066 | 0.5479632116277885 | True   |
+----------+--------------------+--------+


Average planning time over 5 iterations:  0.044 ms.
Average execution time over 5 iterations: 0.542 ms.

+-------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                  |
+-------------------------------------------------------------------------------------------------------------+
| Nested Loop  (cost=0.00..181.20 rows=575 width=17) (actual time=0.124..0.462 rows=1 loops=1)                |
|   ->  Seq Scan on _dict d  (cost=0.00..33.01 rows=1 width=18) (actual time=0.013..0.014 rows=1 loops=1)     |
|   ->  Seq Scan on offers o  (cost=0.00..139.56 rows=575 width=102) (actual time=0.018..0.356 rows=1 loops=1) |
|         Filter: _bdd_has_property(_sentence, 3, 'w43=1'::cstring)                                           |
|         Rows Removed by Filter: 1724                                                                        |
+-------------------------------------------------------------------------------------------------------------+




# ============== insight_6 ============== #

    SELECT AVG(probability) AS certainty_of_the_dataset
    FROM (
        SELECT round(prob(d.dict, o._sentence)::NUMERIC, 4) AS probability
        FROM offers o, _dict d
        WHERE d.name = 'mydict'
    ) AS probabilities;

+--------------------------+
| certainty_of_the_dataset |
+--------------------------+
| 0.70122568115942028986   |
+--------------------------+


Average planning time over 5 iterations:  0.096 ms.
Average execution time over 5 iterations: 102.66 ms.

+-----------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                      |
+-----------------------------------------------------------------------------------------------------------------+
| Aggregate  (cost=202.76..202.77 rows=1 width=32) (actual time=104.198..104.201 rows=1 loops=1)                  |
|   ->  Nested Loop  (cost=0.00..185.51 rows=1725 width=112) (actual time=0.023..1.480 rows=1725 loops=1)         |
|         ->  Seq Scan on _dict d  (cost=0.00..33.01 rows=1 width=18) (actual time=0.021..0.023 rows=1 loops=1)   |
|               Filter: ((name)::text = 'mydict'::text)                                                           |
|         ->  Seq Scan on offers o  (cost=0.00..135.25 rows=1725 width=94) (actual time=0.001..0.201 rows=1725    |
|                                                                                                        loops=1) |
+-----------------------------------------------------------------------------------------------------------------+




# ============== probabilistic_1 ============== #

    SELECT round(prob(d.dict, o._sentence)::NUMERIC, 4) AS probability, o.*
    FROM offers o, _dict d
    WHERE d.name = 'mydict'
    ORDER BY probability DESC;
```

```
+-------------+----+------------+-------+-------+----------+-------------+-------+--------------+-----+
| probability | id | cluster_id | title | brand | category | description | price | identifiers  | ... |
+-------------+----+------------+-------+-------+----------+-------------+-------+--------------+-----+
|             |    |            |       |       |          |             |       |              |     |
+-------------+----+------------+-------+-------+----------+-------------+-------+--------------+-----+
The first 20 out of 1725 rows are shown.
Some returned records were too large to display. This query returned 1725 rows.


Average planning time over 5 iterations:  0.097 ms.
Average execution time over 5 iterations: 102.78 ms.

+-------------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                        |
+-------------------------------------------------------------------------------------------------------------------+
| Sort  (cost=291.19..295.50 rows=1725 width=543) (actual time=104.401..104.465 rows=1725 loops=1)                  |
|   Sort Key: (round((prob(d.dict, o._sentence))::numeric, 4)) DESC                                                 |
|   Sort Method: quicksort  Memory: 1007kB                                                                          |
|   ->  Nested Loop  (cost=0.00..198.45 rows=1725 width=543) (actual time=0.081..103.465 rows=1725 loops=1)         |
|         ->  Seq Scan on _dict d  (cost=0.00..33.01 rows=1 width=18) (actual time=0.021..0.023 rows=1 loops=1)     |
|               Filter: ((name)::text = 'mydict'::text)                                                             |
|         ->  Seq Scan on offers o  (cost=0.00..135.25 rows=1725 width=511) (actual time=0.001..0.175 rows=1725     |
|                                                                                                          loops=1) |
+-------------------------------------------------------------------------------------------------------------------+



# ============== probabilistic_2 ============== #

    SELECT category, SUM(prob(d.dict, o._sentence)) AS expected_count
    FROM offers o, _dict d
    WHERE d.name = 'mydict'
    GROUP BY category
    ORDER BY expected_count DESC;

+----------------------------+--------------------+
| category                   | expected_count     |
+----------------------------+--------------------+
| Tools_and_Home_Improvement | 151.76652960050382 |
| Home_and_Garden            | 132.46422637057302 |
| Clothing                   | 105.81948233534979 |
| Automotive                 | 99.07152758271758  |
| Office_Products            | 97.99577598105066  |
| Other_Electronics          | 72.35240584584147  |
| Books                      | 50.013460503500035 |
| Sports_and_Outdoors        | 48.177061794882086 |
| Computers_and_Accessories  | 46.844157965517795 |
| Jewelry                    | 45.249903459438904 |
| Shoes                      | 44.811806149479914 |
| Health_and_Beauty          | 42.40316057105195  |
| Grocery_and_Gourmet_Food   | 37.08972946966475  |
| Musical_Instruments        | 31.464575133557027 |
| Luggage_and_Travel_Gear    | 29.667148937025335 |
| Camera_and_Photo           | 29.112691031904887 |
| Toys_and_Games             | 28.575616637317314 |
| CDs_and_Vinyl              | 23.75262078226958  |
| not found                  | 23.023654480188345 |
+----------------------------+--------------------+
The first 20 out of 25 rows are shown.


Average planning time over 5 iterations:  0.079 ms.
Average execution time over 5 iterations: 101.2 ms.

+-------------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                        |
+-------------------------------------------------------------------------------------------------------------------+
| Sort  (cost=199.28..199.34 rows=25 width=24) (actual time=103.790..103.792 rows=25 loops=1)                       |
|   Sort Key: (sum(prob(d.dict, o._sentence))) DESC                                                                 |
|   Sort Method: quicksort  Memory: 26kB                                                                            |
|   ->  HashAggregate  (cost=198.45..198.70 rows=25 width=24) (actual time=103.776..103.780 rows=25 loops=1)        |
|         Group Key: o.category                                                                                     |
|         Batches: 1  Memory Usage: 24kB                                                                            |
|         ->  Nested Loop  (cost=0.00..185.51 rows=1725 width=128) (actual time=0.017..0.952 rows=1725 loops=1)     |
|               ->  Seq Scan on _dict d  (cost=0.00..33.01 rows=1 width=18) (actual time=0.015..0.016 rows=1        |
|                                                                                                          loops=1) |
|                     Filter: ((name)::text = 'mydict'::text)                                                       |
|               ->  Seq Scan on offers o  (cost=0.00..135.25 rows=1725 width=110) (actual time=0.001..0.180         |
|                                                                                                rows=1725 loops=1) |
+-------------------------------------------------------------------------------------------------------------------+
# ============== probabilistic_3 ============== #

    SELECT cluster_id, ROUND(SUM(id * prob(d.dict, o._sentence))::NUMERIC, 2) AS expected_sum, COUNT(id) AS
number_of_offers
    FROM offers o, _dict d
    WHERE d.name = 'mydict'
    GROUP BY cluster_id
    ORDER BY number_of_offers DESC;

+------------+--------------+------------------+
| cluster_id | expected_sum | number_of_offers |
+------------+--------------+------------------+
| 982        | 10738252.63  | 5                |
| 192        | 36393937.33  | 5                |
| 1032       | 4897011.00   | 5                |
| 1228       | 9658046.19   | 5                |
| 150        | 29429384.50  | 5                |
| 89         | 32855493.50  | 5                |
| 1227       | 10775660.63  | 5                |
| 102        | 31057848.43  | 5                |
| 948        | 8731025.46   | 5                |
| 1162       | 12397788.80  | 5                |
| 996        | 9219473.40   | 5                |
| 1215       | 6089245.46   | 5                |
| 993        | 6451651.83   | 5                |
| 77         | 48614400.50  | 5                |
| 968        | 6144958.00   | 5                |
| 1046       | 3255967.20   | 5                |
| 64         | 41167332.50  | 5                |
| 1103       | 11885774.13  | 5                |
| 131        | 27397223.00  | 5                |
+------------+--------------+------------------+
The first 20 out of 1111 rows are shown.


Average planning time over 5 iterations:  0.09 ms.
Average execution time over 5 iterations: 103.02 ms.

+-----------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                      |
+-----------------------------------------------------------------------------------------------------------------+
| Sort  (cost=284.26..287.03 rows=1111 width=48) (actual time=103.266..103.300 rows=1111 loops=1)                 |
|   Sort Key: (count(o.id)) DESC                                                                                  |
|   Sort Method: quicksort  Memory: 120kB                                                                         |
|   ->  HashAggregate  (cost=211.39..228.05 rows=1111 width=48) (actual time=102.531..103.146 rows=1111 loops=1)  |
|         Group Key: o.cluster_id                                                                                 |
|         Batches: 1  Memory Usage: 193kB                                                                         |
|         ->  Nested Loop  (cost=0.00..185.51 rows=1725 width=128) (actual time=0.029..1.107 rows=1725 loops=1)   |
|               ->  Seq Scan on _dict d  (cost=0.00..33.01 rows=1 width=18) (actual time=0.025..0.026 rows=1      |
|                                                                                                        loops=1) |
|                     Filter: ((name)::text = 'mydict'::text)                                                     |
|               ->  Seq Scan on offers o  (cost=0.00..135.25 rows=1725 width=110) (actual time=0.002..0.176       |
|                                                                                              rows=1725 loops=1) |
+-----------------------------------------------------------------------------------------------------------------+




# ============== probabilistic_4 ============== #

    WITH category_sentence AS (
        SELECT timeout, category, AGG_OR(_sentence) AS sentence
        FROM offers
        GROUP BY category
    )
    SELECT cs.*, round(prob(d.dict, cs.sentence)::NUMERIC, 4) AS probability
    FROM category_sentence cs, _dict d
    WHERE d.name = 'mydict'
    ORDER BY probability ASC;


The following error occurred while executing this query:
column "timeout" does not exist
LINE 3:         SELECT timeout, category, AGG_OR(_sentence) AS sente...
                       ^
```

```
# ============== probabilistic_5 ============== #

    SELECT o.id, o.cluster_id, o.brand, o.category, o.identifiers, round(prob(d.dict, _sentence)::NUMERIC, 4) AS
probability
    FROM offers o, _dict d
    WHERE cluster_id IN (
        SELECT cluster_id
        FROM offers
        WHERE title LIKE '%card%'
        OR description LIKE '%card%'
    )
    ORDER BY probability DESC
    LIMIT 1;

+----------+------------+-------+----------------------------+------------------------------+-------------+
| id       | cluster_id | brand | category                   | identifiers                  | probability |
+----------+------------+-------+----------------------------+------------------------------+-------------+
| 16951969 | 97         | None  | Tools_and_Home_Improvement | [{'/sku': '[23003393usen]'}] | 1.0000      |
+----------+------------+-------+----------------------------+------------------------------+-------------+


Average planning time over 5 iterations:  0.47 ms.
Average execution time over 5 iterations: 46.812 ms.

+--------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                   |
+--------------------------------------------------------------------------------------------------------------+
| Limit  (cost=325.77..325.77 rows=1 width=105) (actual time=46.124..46.126 rows=1 loops=1)                    |
|   ->  Sort  (cost=325.77..326.28 rows=205 width=105) (actual time=46.123..46.125 rows=1 loops=1)             |
|         Sort Key: (round((prob(d.dict, o._sentence))::numeric, 4)) DESC                                      |
|         Sort Method: top-N heapsort  Memory: 25kB                                                            |
|         ->  Nested Loop  (cost=145.06..324.74 rows=205 width=105) (actual time=4.937..46.088 rows=129 loops=1) |
|               ->  Seq Scan on _dict d  (cost=0.00..33.01 rows=1 width=18) (actual time=0.029..0.029 rows=1   |
|                                                                                                     loops=1) |
|               ->  Hash Semi Join  (cost=145.06..288.15 rows=205 width=167) (actual time=1.345..1.709 rows=129 |
|                                                                                                     loops=1) |
|                     Hash Cond: (o.cluster_id = offers.cluster_id)                                            |
|                     ->  Seq Scan on offers o  (cost=0.00..135.25 rows=1725 width=167) (actual time=0.002..0.111 |
|                                                                                           rows=1725 loops=1) |
|                     ->  Hash  (cost=143.88..143.88 rows=95 width=8) (actual time=1.334..1.334 rows=76 loops=1) |
|                           Buckets: 1024  Batches: 1  Memory Usage: 11kB                                      |
|                           ->  Seq Scan on offers  (cost=0.00..143.88 rows=95 width=8) (actual time=0.052..1.324 |
|                                                                                             rows=76 loops=1) |
|                                 Filter: ((title ~~ '%card%'::text) OR (description ~~ '%card%'::text))       |
|                                 Rows Removed by Filter: 1649                                                 |
+--------------------------------------------------------------------------------------------------------------+



# ============== probabilistic_6 ============== #

    SELECT o.*
    FROM offers o, _dict d
    WHERE title LIKE '%card%'
       OR description LIKE '%card%'
    AND prob(d.dict, _sentence) > 0.45
    AND prob(d.dict, _sentence) < 0.55;

+----+------------+-------+-------+----------+-------------+-------+---------------+-----+
| id | cluster_id | title | brand | category | description | price | identifiers   | ... |
+----+------------+-------+-------+----------+-------------+-------+---------------+-----+
|    |            |       |       |          |             |       |               |     |
+----+------------+-------+-------+----------+-------------+-------+---------------+-----+
The first 20 out of 43 rows are shown.
Some returned records were too large to display. This query returned 43 rows.


Average planning time over 5 iterations:  0.579 ms.
Average execution time over 5 iterations: 11.178 ms.

+--------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                   |
+--------------------------------------------------------------------------------------------------------------+
| Nested Loop  (cost=0.00..179.26 rows=29 width=511) (actual time=0.029..10.838 rows=43 loops=1)               |
|   Join Filter: ((o.title ~~ '%card%'::text) OR ((o.description ~~ '%card%'::text) AND (prob(d.dict, o._sentence) |
|                       > '0.45'::double precision) AND (prob(d.dict, o._sentence) < '0.55'::double precision))) |
|   Rows Removed by Join Filter: 33                                                                            |
|   ->  Seq Scan on _dict d  (cost=0.00..33.01 rows=1 width=18) (actual time=0.007..0.008 rows=1 loops=1)      |
|   ->  Seq Scan on offers o  (cost=0.00..143.88 rows=95 width=511) (actual time=0.021..0.964 rows=76 loops=1) |
|         Filter: ((title ~~ '%card%'::text) OR (description ~~ '%card%'::text))                               |
|         Rows Removed by Filter: 1649                                                                         |
+--------------------------------------------------------------------------------------------------------------+
      # ============================================= #
      # ==============   QuestionMark  ============== #
      # ============================================= #
      # The query results file.
      # Run on DuBio.

This file contains the query results and runtimes of this benchmark test.
The query plan and average run time are produced by PostgreSQL EXPLAIN ANALYSE.
Please see 'QM_metrics_results' for the results of the metrics.


# ============== IUD_1_rollback ============== #

    INSERT INTO offers (id, cluster_id, title, brand, category, description, price, identifiers, keyvaluepairs,
spectablecontent, "_sentence")
        VALUES(-464, 77, ..., Bdd('b77x1=1&v77=1') ),
              (-466, 77, ..., Bdd('b78x1=0&v78=1') ),
              (-468, 77, ..., Bdd('b77x1=2&v77=1') ),
              (-469, 77, ..., Bdd('b78x1=1&v78=1') ),
              (-471, 77, ..., Bdd('b77x1=0&v77=1') );

    UPDATE _dict
    SET dict = add(dict, 'b77x1=0:0.24454, b77x1=1:0.42911, b77x1=2:0.32635, b78x1=0:0.5, b78x1=1:0.5, v77=1:0.629,
                          v77=2:0.125, v77=3:0.246')
    WHERE name='mydict';


Average planning time over 5 iterations:  0.023 ms.
Average execution time over 5 iterations: 0.295 ms.

+---------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                    |
+---------------------------------------------------------------------------------------------------------------+
| Insert on offers  (cost=0.00..0.06 rows=0 width=0) (actual time=0.311..0.311 rows=0 loops=1)                  |
|   ->  Values Scan on "*VALUES*"  (cost=0.00..0.06 rows=5 width=304) (actual time=0.001..0.014 rows=5 loops=1) |
+---------------------------------------------------------------------------------------------------------------+


Average planning time over 5 iterations:  0.067 ms.
Average execution time over 5 iterations: 2.298 ms.

+-------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                            |
+-------------------------------------------------------------------------------------------------------+
| Update on _dict  (cost=0.00..33.02 rows=0 width=0) (actual time=2.178..2.179 rows=0 loops=1)          |
|   ->  Seq Scan on _dict  (cost=0.00..33.02 rows=1 width=38) (actual time=1.029..1.030 rows=1 loops=1) |
|         Filter: ((name)::text = 'mydict'::text)                                                       |
+-------------------------------------------------------------------------------------------------------+



# ============== IUD_2_rollback ============== #

    INSERT INTO offers(id, cluster_id, title, brand, category, description, price, identifiers, keyvaluepairs,
                       spectablecontent, _sentence)
        SELECT * FROM bulk_insert;

    UPDATE _dict
    SET dict = add(dict, 'b000x1=0:0.500000, b000x1=1:0.500000, ..., v966=1:0.632582, v966=2:0.203147')
    WHERE name='mydict';


Average planning time over 5 iterations:  0.046 ms.
Average execution time over 5 iterations: 4.034 ms.

+------------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                       |
+------------------------------------------------------------------------------------------------------------------+
| Insert on offers  (cost=0.00..76.00 rows=0 width=0) (actual time=3.372..3.373 rows=0 loops=1)                    |
|  ->  Seq Scan on bulk_insert  (cost=0.00..76.00 rows=1000 width=475) (actual time=0.005..0.188 rows=1000 loops=1)|
+------------------------------------------------------------------------------------------------------------------+

Average planning time over 5 iterations:  0.069 ms.
Average execution time over 5 iterations: 4.908 ms.

+-------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                            |
+-------------------------------------------------------------------------------------------------------+
| Update on _dict  (cost=0.00..33.02 rows=0 width=0) (actual time=4.875..4.876 rows=0 loops=1)          |
|   ->  Seq Scan on _dict  (cost=0.00..33.02 rows=1 width=38) (actual time=3.237..3.237 rows=1 loops=1) |
|         Filter: ((name)::text = 'mydict'::text)                                                       |
+-------------------------------------------------------------------------------------------------------+
# ============== IUD_3_rollback ============== #

    UPDATE _dict
    SET dict = upd(dict, 'a7x1=0:0.3992, a7x1=1:0.6008, ..., w8=2:0.329, w8=3:0.184')
    WHERE name='mydict';


Average planning time over 5 iterations:  0.041 ms.
Average execution time over 5 iterations: 1.128 ms.

+-------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                            |
+-------------------------------------------------------------------------------------------------------+
| Update on _dict  (cost=0.00..33.02 rows=0 width=0) (actual time=1.018..1.019 rows=0 loops=1)          |
|   ->  Seq Scan on _dict  (cost=0.00..33.02 rows=1 width=38) (actual time=0.090..0.091 rows=1 loops=1) |
|         Filter: ((name)::text = 'mydict'::text)                                                       |
+-------------------------------------------------------------------------------------------------------+




# ============== IUD_4_rollback ============== #

    WITH max_cluster AS (
        SELECT (max(cluster_id) + 1) AS max_id
        FROM offers
    )
    UPDATE offers
    SET cluster_id = max_cluster.max_id,
        _sentence = Bdd('1')
    FROM max_cluster
    WHERE id = 2689021;

    WITH max_cluster AS (
        SELECT max(cluster_id) + 1 AS max_id
        FROM offers
    )
    UPDATE offers
    SET cluster_id = max_cluster.max_id,
        _sentence = Bdd('1')
    FROM max_cluster
    WHERE id = 7257664;

    UPDATE offers
    SET _sentence = Bdd('a162x5=0&w162=0')
    WHERE id = 10198975;

    UPDATE offers
    SET _sentence = Bdd('a162x5=1&w162=0')
    WHERE id = 2668263;

    UPDATE _dict
    SET dict = add(dict, 'w162=0:0.83')
    WHERE name='mydict';

    UPDATE _dict
    SET dict = del(dict, 'a162x5=2')
    WHERE name='mydict';


Average planning time over 5 iterations:  0.076 ms.
Average execution time over 5 iterations: 0.998 ms.

+------------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                       |
+------------------------------------------------------------------------------------------------------------------+
| Update on offers  (cost=670.61..1341.26 rows=0 width=0) (actual time=1.101..1.102 rows=0 loops=1)                |
|   ->  Nested Loop  (cost=670.61..1341.26 rows=1 width=78) (actual time=0.758..0.976 rows=1 loops=1)              |
|         ->  Seq Scan on offers  (cost=0.00..670.61 rows=1 width=6) (actual time=0.047..0.263 rows=1 loops=1)     |
|               Filter: (id = 2689021)                                                                             |
|               Rows Removed by Filter: 1724                                                                       |
|         ->  Subquery Scan on max_cluster  (cost=670.61..670.63 rows=1 width=40) (actual time=0.710..0.710 rows=1 |
|                                                                                                         loops=1) |
|               ->  Aggregate  (cost=670.61..670.62 rows=1 width=8) (actual time=0.707..0.707 rows=1 loops=1)      |
|                     ->  Seq Scan on offers offers_1  (cost=0.00..649.89 rows=8289 width=8) (actual               |
|                                                                             time=0.001..0.533 rows=1725 loops=1) |
+------------------------------------------------------------------------------------------------------------------+
Average planning time over 5 iterations:  0.062 ms.
Average execution time over 5 iterations: 0.84 ms.

+-----------------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                                      |
+-----------------------------------------------------------------------------------------------------------------+
| Update on offers  (cost=671.79..1343.61 rows=0 width=0) (actual time=1.153..1.154 rows=0 loops=1)               |
|   ->  Nested Loop  (cost=671.79..1343.61 rows=1 width=78) (actual time=0.857..1.138 rows=1 loops=1)             |
|         ->  Seq Scan on offers  (cost=0.00..671.79 rows=1 width=6) (actual time=0.050..0.330 rows=1 loops=1)    |
|               Filter: (id = 7257664)                                                                            |
|               Rows Removed by Filter: 1724                                                                      |
|         ->  Subquery Scan on max_cluster  (cost=671.79..671.81 rows=1 width=40) (actual time=0.805..0.806 rows=1 |
|                                                                                                        loops=1) |
|               ->  Aggregate  (cost=671.79..671.80 rows=1 width=8) (actual time=0.802..0.802 rows=1 loops=1)     |
|                     ->  Seq Scan on offers offers_1  (cost=0.00..651.03 rows=8303 width=8) (actual              |
|                                                                          time=0.001..0.602 rows=1725 loops=1) |
+-----------------------------------------------------------------------------------------------------------------+


Average planning time over 5 iterations:  0.031 ms.
Average execution time over 5 iterations: 0.727 ms.

+--------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                             |
+--------------------------------------------------------------------------------------------------------+
| Update on offers  (cost=0.00..672.98 rows=0 width=0) (actual time=1.293..1.294 rows=0 loops=1)         |
|   ->  Seq Scan on offers  (cost=0.00..672.98 rows=1 width=38) (actual time=0.067..1.131 rows=1 loops=1) |
|         Filter: (id = 10198975)                                                                        |
|         Rows Removed by Filter: 1724                                                                   |
+--------------------------------------------------------------------------------------------------------+


Average planning time over 5 iterations:  0.056 ms.
Average execution time over 5 iterations: 1.138 ms.

+--------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                             |
+--------------------------------------------------------------------------------------------------------+
| Update on offers  (cost=0.00..672.98 rows=0 width=0) (actual time=0.985..0.985 rows=0 loops=1)         |
|   ->  Seq Scan on offers  (cost=0.00..672.98 rows=1 width=38) (actual time=0.064..0.970 rows=1 loops=1) |
|         Filter: (id = 2668263)                                                                         |
|         Rows Removed by Filter: 1724                                                                   |
+--------------------------------------------------------------------------------------------------------+


Average planning time over 5 iterations:  0.055 ms.
Average execution time over 5 iterations: 1.313 ms.

+------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                           |
+------------------------------------------------------------------------------------------------------+
| Update on _dict  (cost=0.00..33.02 rows=0 width=0) (actual time=1.132..1.132 rows=0 loops=1)         |
|   ->  Seq Scan on _dict  (cost=0.00..33.02 rows=1 width=38) (actual time=0.111..0.112 rows=1 loops=1) |
|         Filter: ((name)::text = 'mydict'::text)                                                      |
+------------------------------------------------------------------------------------------------------+


Average planning time over 5 iterations:  0.043 ms.
Average execution time over 5 iterations: 1.104 ms.

+------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                           |
+------------------------------------------------------------------------------------------------------+
| Update on _dict  (cost=0.00..33.02 rows=0 width=0) (actual time=1.070..1.070 rows=0 loops=1)         |
|   ->  Seq Scan on _dict  (cost=0.00..33.02 rows=1 width=38) (actual time=0.081..0.082 rows=1 loops=1) |
|         Filter: ((name)::text = 'mydict'::text)                                                      |
+------------------------------------------------------------------------------------------------------+
```

```
# ============= IUD_5_rollback ============= #

    DELETE FROM offers
    WHERE cluster_id = 41;

    UPDATE _dict
    SET dict = del(dict, 'a41x1=0, a41x1=1, ..., w44=4, w44=5')
    WHERE name='mydict';
```

```
Average planning time over 5 iterations:  0.019 ms.
Average execution time over 5 iterations: 0.428 ms.

+---------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                              |
+---------------------------------------------------------------------------------------------------------+
| Delete on offers  (cost=0.00..672.98 rows=0 width=0) (actual time=0.273..0.273 rows=0 loops=1)          |
|   ->  Seq Scan on offers  (cost=0.00..672.98 rows=24 width=6) (actual time=0.007..0.268 rows=5 loops=1) |
|         Filter: (cluster_id = 41)                                                                       |
|         Rows Removed by Filter: 1720                                                                    |
+---------------------------------------------------------------------------------------------------------+
```

```
Average planning time over 5 iterations:  0.037 ms.
Average execution time over 5 iterations: 1.194 ms.

+-------------------------------------------------------------------------------------------------------+
| QUERY PLAN                                                                                            |
+-------------------------------------------------------------------------------------------------------+
| Update on _dict  (cost=0.00..33.02 rows=0 width=0) (actual time=1.037..1.037 rows=0 loops=1)          |
|   ->  Seq Scan on _dict  (cost=0.00..33.02 rows=1 width=38) (actual time=0.092..0.093 rows=1 loops=1) |
|         Filter: ((name)::text = 'mydict'::text)                                                       |
+-------------------------------------------------------------------------------------------------------+
```