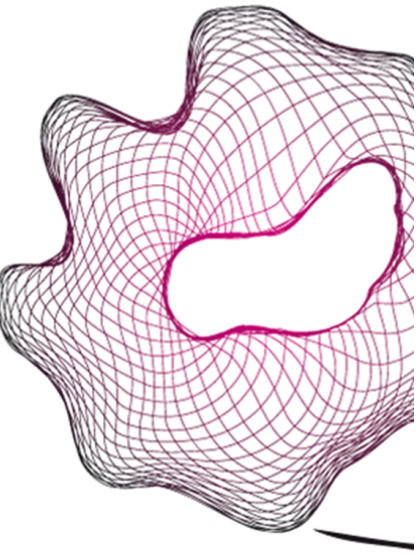
UNIVERSITY OF TWENTE.

**Faculty of Electrical Engineering,
Mathematics & Computer Science**



- Research Topics -

Designing a benchmark for probabilistic databases

Nikki Zandbergen

M.Sc. Thesis
August 2022

Supervisor:
dr. ir. M. van Keulen

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

Designing a benchmark for probabilistic databases

Nikki Zandbergen
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
n.zandbergen@student.utwente.nl

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 robust enough to function in a real-world environment. This research will contribute to the development of novel probabilistic databases by developing a benchmark specifically designed for real-world strain testing of probabilistic databases. This benchmark will be used to evaluate the performance and functionality of the state-of-the-art probabilistic database MayBMS and the novel probabilistic database DuBio. With these results, it can be evaluated what research still needs to be conducted before probabilistic databases can expect real-world use.

Keywords

Benchmark, Probabilistic Database, DuBio, MayBMS, Deduplication, Product Matching, Entity Resolution.

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. While a large part of the data produced today is incomplete or uncertain, it is often still treated in a deterministic manner.

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 [15]. Probabilistic data processing can aid decisions in more scientific areas, such as bio-informatics [61] and healthcare [41], but also adds value in various business processes, which rely on decisions based on data from different sources. To get this data ready for 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 [15]. 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 [35]. Considering probabilistic databases, a benchmark should provide a dataset and a set of queries to test the systems. Benchmarking also provides a fair comparison of two different systems. While many benchmarks are available for deterministic databases, such as TPC and SPEC [35], 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, 11, 26, 33, 48], scientific data collection [5, 31, 45, 64], data integration (data deduplication) [10, 17, 56, 32, 40, 45, 49], user profiling [64], medical data [8, 31, 38] and human-entered data [5]. 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 [17]. 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 [45].

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 [26, 48]. 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 [17]. Unfortunately, no probabilistic database management system to date performs well enough to be used in various real-world scenarios [60].

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 [57] and the state-of-the-art probabilistic database MayBMS [7] will be

Research Topics. August 8, 2022, Enschede, The Netherlands.

tested. The benchmark will also be run with the deterministic DBMS PostgreSQL [52] to provide a baseline performance.

During this research there will be a special interest in product matching. Product matching is a form of entity resolution [32], which refers to the process of identifying which data entities from multiple sources refer to the same real-world entity [42, 66]. Entity resolution aids the goal of data integration, which should lead to an increased data quality and size usable for further analysis [66].

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 [42]. 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 two or more systems with overlapping information. Having a system that can translate this uncertainty and show it to the user of the system can enhance business intelligence decisions and tackle the issues discussed earlier.

The rest of this paper is structured as follows. In Section 2 an overview will be given on previous work on probabilistic databases and benchmarking. Section 3 introduces the systems that will be used for this research. Section 4 discusses the benchmark dataset for running experiments with the selected technologies. In Section 5 the purpose and goals of this research are listed, and in section 6 the proposed methodology for this research is discussed. Section 7 provides the initial results of this research and shows that the proposed research is feasible. In Section 8 the work programme for the final research is listed and a planning is provided.

2. BACKGROUND

In this section, the definition of a probabilistic database will be given and a background on probabilistic databases will be provided. Additionally, previous work on benchmarking technologies for databases will be discussed and the shortcomings of those will be evaluated.

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 [10]. Probabilistic databases are systems that store uncertain data and support complex queries that translate this uncertainty to the user [16]. 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 [16]. 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 [29].

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 [62]. 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 [49].

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 [4, 31, 62].

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 [28].
$$\langle R_1^i, \ldots, R_k^i, p^{[i]} \rangle \in W$$
- *Probabilistic database:* A probabilistic database is a finite set of structures, where each set of relations within the structure has a valid probability [28].
$$W = \{\langle R_1^1, \ldots, R_k^1, p^{[1]} \rangle, \ldots, \langle R_1^n, \ldots, R_k^n, p^{[n]} \rangle\}$$
$$where \sum_{1 \leq i \leq n} p^{[i]} = 1.$$
- *Descriptive sentence:* A descriptive sentence describes a subset of possible worlds using the notation of set theory [58]. 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 [21, 24], 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 [33]. 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 [33]. Although earlier research identified several ways in which uncertain data could be modelled [17], 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 [49, 61].

Over the years, some of the more promising probabilistic databases developed are MayBMS [5], MCDB (Monte Carlo Database) [25], and Trio [64]. 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 [55]. 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 [36].

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 [13, 14]. 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. For a benchmark to be fair, it should work indiscriminately [28]. 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 [39].

Having a benchmark for testing technologies helps both the developers and the consumers of a technology [35]. For developers, the results of a benchmark test can be used to check for areas where the technology can be improved and to show the strengths of the technology to stakeholders and potential customers. For consumers, it provides a standardised way to compare various technologies. Benchmarking additionally provides convenience in testing technologies. Without benchmarking, a technical specialist should be hired to test a system thoroughly, which would cost a business many resources in terms of time and money [38].

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. An important aspect is repeating the tests to obtain a more reliable performance estimation. More iterations lead to more reliable results, but often five to ten iterations is performed [20]. One way to design a benchmark is by following design science.

Numerous benchmark tools are already available for testing deterministic database management systems. These include Apache JMeter, BAPco, SPEC, TPC and Wisconsin [20, 35], 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 [53].

For probabilistic databases, no standard benchmark tools are available yet [54]. 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 [47]. MayBMS created an adaptation of the TPC-H benchmark to test their system, called Probabilistic TPC-H [31]. Other researches also designed their own benchmark to test MayBMS, such as [12] and [54]. 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.

3. TECHNOLOGIES

For this research, three relational database management systems will be used and put to the test with the designed benchmark. In this section, each of these systems will be introduced, and their identified strengths and weaknesses from earlier research will be discussed.

3.1 PostgreSQL

PostgreSQL [52] 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 [51]. 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 [50].

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 [52]. 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 [53]. Hence why many projects, including MayBMS and DuBio, run on top of PostgreSQL.

3.2 MayBMS

MayBMS [7] 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 [5]. Their aim is to have a robust database system that could be used in real applications [28]. MayBMS is developed on top of PostgreSQL in a way to ensure a fully integrated system [5, 28]. 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 [31].

The internal functioning of MayBMS is based on possible worlds theory [5] and uses U-relational databases [4]. 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 [28]. Because of this design, MayBMS supports complex dependencies [36].

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. [6].

Experiments conducted by the developers of MayBMS showed that the system can fulfil the expectations of being a usable probabilistic DMBS. These experiments showed a runtime execution close to that of conventional query evaluation [5] and suggest that MayBMS will perform well in real-world scenarios [8, 31].

Various researches investigating the usefulness and opportunities of probabilistic databases have used MayBMS as a prototype. [3] and [38] used MayBMS for the purpose of detecting faulty sensors and modelling patient counts respectively. Both researches found MayBMS to be of great use. Also [61] 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 [49, 61]. This issue is due to a restriction from PostgreSQL on the maximum amount of columns per table [52], which will be reached when constructing large and complex sentences. A maximum of five hundred random variable assignments can be supported per record [61].

2. MayBMS cannot handle OR-relations in sentences [5]. 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 [49].

4. The performance of MayBMS is not always stable. [12] noted that MayBMS reported errors and memory issues on certain runs on random data, especially when data sets grew over 1 million records. [49] reported a rapid growth of runtime when facing larger datasets.

5. For the purpose of the research of [49], MayBMS did not work out-of-the-box and had to be tweaked to support relations of arbitrary arity and provide a fair timing.

3.3 DuBio

DuBio [57] 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 [15], making it less dependable on a specific PostgreSQL version and allowing it to run on several versions, including the currently latest version PostgreSQL 14.4.

The internal functioning of DuBio is also based on the possible worlds theory [60] and focusses on record-level uncertainty. DuBio represents this uncertainty using a type of Binary Decision Diagrams [15]. DuBio tries to address issues found in other probabilistic database management systems to deliver a system usable for real-world applications [60]. 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 [15]. 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 [23].

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 [15, 59], although development is still in progress to improve the time efficiency of complex queries even further.

4. DATASET

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 this dataset are laid down. A selection of datasets is evaluated to these requirements and the dataset deemed most fit is selected. Finally, an elaboration on the chosen dataset is provided.

4.1 Requirements Specification

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 uncertainty 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.

4.2 Dataset Evaluation

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, 18, 34], and existing datasets, as used in [37, 46, 66, 67].

Regarding the self-collected datasets, it is 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 ourselves for the purpose of this research is not feasible.

From the existing datasets, the Web Data Commons dataset was most frequently used [46, 66, 67]. Other datasets found in the selection of researches include the Yahoo's Gemini Product Ads dataset [46], UCI datasets [37], and the LEAPME-dataset [9].

When evaluating these datasets according to our 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 [27] were also evaluated, but none were deemed more fit than the WDC dataset.

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 Chosen Dataset

For this research, the WDC Product Data Corpus and Gold Standard for Large-Scale Product Matching, Version 2.0 [63] will be used.

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, they also offer an English language subset. This subset consists of 16 million product offers [44]. Version 2.0 of this dataset was released in 2020. It contains the same data as version 1.0, but has a simplified identification scheme [63].

The product offers in the dataset are also clustered by Web Data Commons. The 16 million product offers 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 thousand clusters with size 3 or 4. Clusters of a size greater than 80 are filtered out of the dataset, as these are likely noise [44].

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 [44]. This part of the dataset is particularly interesting, as it can aid as a ground truth during the performance comparison of the various (probabilistic) relational database technologies.

5. RESEARCH PURPOSE

The goal of this research is to design a benchmark with the purpose of verifying the performance and scalability of probabilistic databases. The benchmark will be designed with real-world cases in mind, trying to get probabilistic database technologies out of the scientific world and into everyday applications. When the benchmark is designed, the performance and scalability of the novel probabilistic database DuBio and the state-of-the-art MayBMS will be evaluated with this benchmark. The performance of the deterministic DBMS PostgreSQL will be used as a baseline. The experiments run with PostgreSQL will also guide to show what the added value of using a probabilistic database system is in the context of the chosen application scenario.

For this research, the following research question will be answered:

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

The following sub-questions will be answered to guide in answering the main question:

SQ1 What real-world scenario should the benchmark measure?

SQ2 How can a realistic probability model be designed to express doubt in an uncertain dataset?

SQ3 What queries are representable for real-world use of such a dataset for probabilistic databases?

SQ4 How do the novel probabilistic database DuBio and the state-of-the-art MayBMS perform when benchmarking these technologies with the developed benchmark?

SQ5 How much added value does the use of probabilistic databases provide, compared to deterministic databases?

SQ6 What steps are still necessary to achieve widespread use of probabilistic databases in everyday applications?

6. METHODOLOGY AND APPROACH

This section describes the methodology that will be used to answer the research questions defined in section 5. The research required for answering these question can broadly be divided into literature research, designing and quantitative research. The research consists of two phases: designing the benchmark and benchmarking the specified technologies.

6.1 Designing the benchmark

Designing the benchmark is part of SQ1, SQ2 and SQ3. This phase will largely be done by performing literature research and designing. For answering each of these questions, the design science philosophy will be used.

From the book of [65] and the paper of [43], 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 will be 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 [22]. This results in a framework fit to solve problems at the intersection of business and IT [43]. Design science research is developed for large multi-paper research [22]. For this research, an adapted version is used for smaller research.

The approach used in this paper will be based on the standard design science methodology, as described in [22], [43] and [65]. As no consensus is yet reached on an exact methodology, the phases will be used as a guideline and adapted where it seems fit. The phases that will be followed are:

1. Problem identification and motivation.
2. Define solution objectives.
3. Design and development.
4. Demonstration.
5. Evaluation.
6. Communication.

Phase 1 and 2 have largely been answered in this paper already, but could be enhanced later. SQ1 will be the first step in designing the benchmark for phase 3. It will consist of performing literature review to identify the aspects that should be covered by the benchmark.

SQ2 designs the dataset that the benchmark will use. In this report, the dataset that will be used is defined. However, this dataset is altered to be presented in a deterministic manner. For the benchmark, the dataset needs to be transformed back to a nondeterministic state. Research will be performed to verify how a realistic probability model can be assigned to the present uncertainty in the dataset.

Answering SQ3 consists of two steps. First, literature review will help identify common uses of product matching in the real world and identify several general uses of probabilistic databases that might be interesting to cover with the benchmark. Second, fitting queries will be formulated based on the literature research and additional brainstorming. The insights from SQ1 will also aid in formulating the queries for this sub-question.

From the literature research performed in this document, it can be concluded that enough research is available to answer each of these questions. Depending on the amount of research already conducted on each of these topics, the answer of the questions based on prior research can differ. To ensure the search for literature is as extensive as possible, different research databases will be queried and papers will be requested when not publicly available. Gaps in earlier research will be supplemented with brainstorming.

When all these sub-questions are answered, the final design of the benchmark can be constructed and phase 3 of the design cycle is finalised.

6.2 Benchmarking the technologies

Benchmarking the technologies is part of SQ4. Additionally, SQ5 and SQ6 are answered by the insights that this benchmarking provides.

In order to answer SQ4, DuBio and MayBMS will both be tested with the designed benchmark. The results of these tests will be described and analysed. Conclusions on the performance will then be drawn. This contributes to phase 4 and 5 of the design cycle.

SQ5 will be answered by also running PostgreSQL with the benchmark. By comparing the results both in performance and provided results, the gap between probabilistic databases and deterministic databases can be identified. Important points to discuss here are not only the difference in execution speed, but also that of the results. As the WDC dataset already contains a clustering, it can be verified how the results from this deterministic approach differ from that of the indeterministic approach. The insights that will be obtained from this analysis will be described. This also contributes to phase 4 and 5 of the design cycle, as SQ4 and SQ5 together will aid in both the demonstration and evaluation of the final performance and reliability of the designed benchmark.

Finally, SQ6 will be answered by identifying the points where DuBio could be improved according to the results of the benchmark. Additional literature research will be performed to verify whether the identified areas for improvement have also been identified by earlier research. This step contributes to phase 5 in the design cycle.

Step 6 of the design cycle will be obtained naturally by detailing on the process and conclusions from this research.

The experiments for the final research will be run on two servers of the university. MayBMS runs at a single machine with an Intel Xeon E5-2407 CPU @ 2.20 GHz, quad core, 64 GB RAM, and 4 TB SATA hard drive. DuBio runs at a single machine with an Intel Xeon e5-2403 CPU @ 1.80 GHz, octa core, 64 GB RAM, and a composed 22 TB hard drive of 8 SATA hard drives in a RAID 0 configuration.

The machine that runs MayBMS contains an older installation of the MayBMS software. Therefore, issues caused by updated packages that MayBMS uses are less probable. DuBio is installed on a system that is better optimised for the workload DuBio provides. The differences between the machines should be taken into account when comparing the performance of the two systems with the benchmark test.

7. INITIAL RESULTS

To verify the feasibility of the proposed methodology, some initial experiments were run. These experiments have focused on setting up the dataset and running some simple queries on all three database management systems. The experiments performed in this section are run on a single machine with an Intel Core i5-7200U CPU @ 2.5 GHz, dual core, 8 GB RAM, and BLA. For these experiments the same dataset will be used as for the final research. However, instead of using the full 16 million offers, the dataset will be capped to include only fifty thousand offers.

The installations of MayBMS, DuBio and PostgreSQL each run in their own Docker container. For these experiments, Docker 4.9.0 is used. Each container also runs with their own PostgreSQL installation. MayBMS runs with PostgreSQL 8.3.3, and DuBio and PostgreSQL both run with the current latest PostgreSQL version, PostgreSQL 14.4.

7.1 Dataset Analysis

Before the various database systems can be queried and tested, a thorough understanding of the dataset should be obtained. This section also discusses how the dataset was loaded into the databases.

For this research, the normalised English offers dataset (offers_corpus_english_v2.json.gz) was downloaded from the WDC product offers website [63]. Within the English offers dataset, each offer is represented as a JSON object. An example offer can be found in Appendix A.1. 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.

In order to query the data and test the three database systems, the data needs to be put in a database. To do this, the WDC product offer file was loaded into a Python program, which transforms each offer in the file to a data structure fit to be inserted into the databases. An adaptation was made to the Python program developed by Jan Flokstra, which was created for version 1.0 of the WDC product corpus dataset. The code used can be found at GitHub [19].

To efficiently query the dataset, the structure was changed to split the product offer identifier and the product information in two separate tables. The database schema of the dataset can be found in Appendix A.2.

To get a first impression on the dataset, some deterministic queries were executed. These queries can be found in Appendix A.3. As the dataset used for this first impression contains just a part of the full WDC dataset, the results are purely indicative and a way to showcase what information could be retrieved from the full dataset.

These queries provide us with the following information:

1. *query 5 + 6:* All product offers have a product category, there is a total of 25 product categories.
2. *query 7:* USD is the currency most used, with 3115 occurrences, followed by EUR with 768 occurrences. There are also 692 product offers that do not have a currency specified. There are also numerous product offers that have a random string as currency, such as 'our price'.
3. *query 8 + 9:* There are 6758 product offers with more than one product identifier. There is a total of 8 identifiers, of which /sku and /productID are most used.
4. *query 10 + 11:* There are 858 product offers that share a cluster with another product offer. There are 534 clusters of size 2, 120 clusters of size 3, and 47 of size 4. The largest cluster is 178 product offers large.

7.2 Testing DuBio

To show the feasibility of the final research, some simple probabilistic queries will be performed on both probabilistic database systems and an initial impression on the performance will be provided. The queries used for the experiments with DuBio can be found in Appendix B.1.

For this demonstration, a preliminary database setup is created. For this, the `cluster_id` column will be removed and arbitrary probabilities will be assigned to the product offers. For DuBio, two new tables are created: `pg_wdc_match` containing the division of offers into clusters with their sentences, and `_dict` containing the dictionary with sentences and probabilities. Both tables are filled with dummy data. Dummy data for `pg_wdc_match` was generated using a hash function. This method was chosen, as this provides a pseudorandom distribution of offers over the clusters and disconnects the placement of an offer in the first cluster of that in the second cluster. A total of 512 clusters were created and each product offer was put into two different product clusters and the cluster `-1`, indicating that the offer has no other matches. The division into the two clusters was generated by taking the following steps:

- The product id was taken, appending 'a' for the first cluster and 'b' for the second cluster.
- This is hashed with sha256 and converted to hexadecimal.
- The nine leftmost bits from the three rightmost bytes are taken. This is interpreted as a number and converted to decimal. The resulting number is the cluster.

As one offer cannot occur twice in the same cluster, a check is done. In such a case, a product offer is put in the next cluster. The division of sentences among the products is performed in a similar manner. Query 9 of Appendix B.1.1 took around 6 seconds to run.

For this experiment, cluster 200 was chosen to perform queries on. Cluster 200 contains 183 product offers. Query 4 of Appendix B.1.2 showed the three categories 'Pet_Supplies', 'Sports_and_Outdoors' and 'CDs_and_Vinyl' to be the most likely with 50, 45 and 43 percent likelihood respectively. To test DuBio even further, a second set of sentences is introduced to represent the confidence that either of these options is the correct category. This resulted in a likelihood of 15, 9 and 22 percent respectively, showing that none of the present categories are very likely to be the correct one.

Although this dummy example is simple, it shows that the installation of DuBio works and that various types of queries can be run. The execution time of the select-queries were all well under a second, indicating that the setup does not cause significant lag in performance. Additionally, this shows that the chosen dataset is well fit for use with probabilistic databases.

7.3 Testing MayBMS

For testing the installation of MayBMS, the same experiment is run as with DuBio. The code used for MayBMS can be found in appendix B.2. Since MayBMS runs on top of PostgreSQL 8.3.3, some of the functions used for DuBio cannot be used. Therefore, some alternative functions were used or are self-defined. This makes MayBMS slower even for standard deterministic queries. Adding 150 000 rows to a table with Query 6 of Appendix B.2.1 took 35 seconds to run.

As with DuBio, this example with MayBMS will also run tests with cluster 200. As the same offer division function is used, the clustering over the two databases is the same. This can also be verified by running Queries 1 and 2 of Appendix B.2.2.

When querying the dataset with MayBMS, different results are obtained than with DuBio. When running Queries 4 and 5 of Appendix B.2.2 the most likely categories are not 'Pet_Sup-plies', 'Sports_and_Outdoors' and 'CDs_and_Vinyl', but 'Office_Products', 'Computers_and_Accessories', 'Tools_and_Home_Improvement' and 'Home_and_Garden', each with 6,7% likelihood. Where DuBio provides a likelihood of the category independent of the other categories present, MayBMS gives a probability distribution over all possible answers. This explains why the provided probability by MayBMS is that low, as there are a total of 24 product categories in cluster 200.

Besides different results obtained, it was noted that MayBMS was less intuitive to work with than DuBio. This can be seen by the way Query 4 needs to be altered for MayBMS to run without errors. When referring to a probabilistic view or table in the FROM clause, MayBMS fails to check for this probabilistic trait and throws the error that the `conf()` function is not supported for certain queries. It was indicated that this behaviour was deliberately implemented to avoid unintended side-effects [30]. Thus, the correct way to formulate the query is by including the full original query and transforming this to a probabilistic sub-query in the FROM clause. Nevertheless, this implementation complicates queries significantly adding additional difficulty to the formulation of complex queries.

8. WORK PROGRAMME

There is a total of twenty weeks full-time indicated for working on this graduation thesis. The provisional deadline for writing this thesis is set at 1 February 2023, but this could change when faced with unforeseen challenges. As more than twenty weeks are available, the workload will be divided over twenty five weeks. The planning of this thesis is provided in Appendix C.

In this chart, rough estimations are made of the time span needed to answer the sub-questions of this research. Designing the benchmark will take an estimated thirteen weeks. Benchmarking the three database systems and evaluating gaps in current research will take an estimated twelve weeks.

During the entire time span, attention is paid to finding papers that provide additional substantiation of why this research is of importance and to improving the introduction and background sections.

The final presentation of this thesis will be planned after these 25 weeks.

9. REFERENCES

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

[2] L. Akritidis, A. Fevgas, P. Bozanis, and C. Makris. *A self-verifying clustering approach to unsupervised matching of product titles*, volume 53. Springer Nature B.V., 2020.

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

[4] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and Simple Relational Processing of Uncertain Data. *IEEE 24th International Conference on Data Engineering*, pages 983–992, 2008.

[5] L. Antova, C. Koch, and D. Olteanu. MayBMS: A Possible Worlds Base Management System. 2006.

[6] L. Antova, C. Koch, and D. Olteanu. MayBMS: Managing incomplete information with probabilistic world-set decompositions. *Proceedings - International Conference on Data Engineering*, pages 1479–1480, 2007.

[7] L. Antova, C. Koch, and D. Olteanu. MayBMS. http://maybms.sourceforge.net/, 2008. Accessed 03 May 2022.

[8] L. Antova, C. Koch, and D. Olteanu. $10^{(10^6)}$ worlds and beyond: Efficient representation and processing of incomplete information. *VLDB Journal*, 18(5):1021–1040, 2009.

[9] D. Ayala, I. Hernández, D. Ruiz, and E. Rahm. Multi-source dataset of e-commerce products with attributes for property matching. *Data in Brief*, 41:1–6, 2022.

[10] N. Ayat, R. Akbarinia, H. Afsarmanesh, and P. Valduriez. Entity resolution for probabilistic data. *Information Sciences*, 277:492–511, 2014.

[11] M. Balazinska, A. Deshpande, M. J. Franklin, P. B. Gibbons, J. Gray, S. Nath, M. Hansen, M. Liebhold, A. Szalay, and V. Tao. Data management in the worldwide sensor web. *IEEE Pervasive Computing*, 6(2):30–40, 2007.

[12] K. Booijink. Evaluating the Scalability of MayBMS, a Probabilistic Database Tool. Bachelor's thesis, University of Twente, 2019.

[13] I. I. Ceylan, A. Darwiche, and G. van Den Broeck. Open-world probabilistic databases. *Proceedings of the International Conference on Knowledge Representation and Reasoning*, pages 339–348, 2016.

[14] I. I. Ceylan, A. Darwiche, and G. van Den Broeck. Open-world probabilistic databases: An abridged report. *IJCAI International Joint Conference on Artificial Intelligence*, 0:4796–4800, 2017.

[15] V. Cincotta. Design and Implementation of a Scalable Probabilistic Database System. Master's thesis, Università di Genova, 2019.

[16] N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: Diamonds in the dirt. *Communications of the ACM*, 52(7):86–94, 2009.

[17] N. Dalvi and D. Suciu. Management of Probabilistic Data: Foundations and Challenges. *Proceedings of*

the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pages 1–12, 2007.

[18] M. Dylla, I. Miliaraki, and M. Theobald. A temporal-probabilistic database model for information extraction. *Proceedings of the VLDB Endowment*, 6(14):1810–1821, 2013.

[19] J. Flokstra, M. van Keulen, and N. Zandbergen. wdc data converter. https://github.com/utwente-dmb/wdc_pdb, 2022. Accessed 29 June 2022.

[20] C. J. Gillan, A. Novakovic, A. H. Marshall, M. Shyamsundar, and D. S. Nikolopoulos. Expediting assessments of database performance for streams of respiratory parameters. *Computers in Biology and Medicine*, 100(May):186–195, 2018.

[21] G. Grahne. Dependency Satisfaction In Databases With Incomplete Information. *Proceedings of the Tenth International Conference on Very Large Data Bases*, pages 37 – 45, 1984.

[22] S. Gregor and A. R. Hevner. Positioning and presenting design science research for maximum impact. *MIS Quarterly*, 37(2):337–355, 2013.

[23] J. Groot Roessink. inSQeLto: a Query Language for Probabilistic Databases. Bachelor's thesis, 2021.

[24] T. Imielinski and L. Witold Jr. Incomplete Information And Dependencies In Relational Databases. *Proceedings of the 1983 ACM SIGMOD internation conference on Management of data*, pages 178–184, 1983.

[25] R. Jampani, L. L. Perez, F. Xu, C. Jermaine, M. Wu, and P. J. Haas. MCDB: A monte carlo approach to managing uncertain data. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 687–700, 2008.

[26] A. S. Jumde and N. S. Chaudhari. Query processing techniques in probabilistic databases. *International Conference on Computing, Analytics and Security Trends*, pages 483–488, 2016.

[27] Kaggle Inc. Kaggle. https://www.kaggle.com/datasets, 2010. Accessed 23 May 2022.

[28] C. Koch. MayBMS: A System for Managing Large Uncertain and Probabilistic Databases. pages 1–34, 2009.

[29] C. Koch and D. Olteanu. Conditioning probabilistic databases. *Proceedings of the VLDB Endowment*, 1(1):313–325, 2008.

[30] C. Koch and D. Olteanu. Private communications, 2022.

[31] C. Koch, D. Olteanu, L. Antova, and J. Huang. MayBMS: A Probabilistic Database System, 2009.

[32] H. Köpcke, A. Thor, S. Thomas, and E. Rahm. Tailoring entity resolution for matching product offers. *ACM International Conference Proceeding Series*, pages 545–550, 2012.

[33] L. V. S. Lakshmanan, R. Ross, and V. S. Subrahmanian. ProbView: A Flexible Probabilistic Database System. *ACM Transactions on Database Systems*, 22(3):419 –469, 1997.

[34] J. Li, Z. Dou, Y. Zhu, X. Zuo, and J. R. Wen. Deep cross-platform product matching in e-commerce. *Information Retrieval Journal*, 23(2):136–158, 2020.

[35] A. Makris, K. Tserpes, G. Spiliopoulos, D. Zissis, and D. Anagnostopoulos. MongoDB Vs PostgreSQL: a comparative study on performance aspects. *GeoInformatica*, 25(1):241–242, 2021.

[36] R. R. Mauritz, F. P. J. Nijweide, J. Goseling, and M. van Keulen. A probabilistic database approach to autoencoder-based cleaning. *ACM Journal of Data and Information Quality, Special Issue on Deep Learning for Data Quality*, jun 2021.

[37] B. Mozafari, P. Sarkar, M. M. Franklin, M. Jordan, and S. Madden. Scaling up crowdsourcing to very large datasets: A case for active learning. *Proceedings of the VLDB Endowment*, 8(2):125–136, 2014.

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

[39] Oxford Languages. *Oxford Dictionary of English*. Oxford University Press, 3 edition, 2010.

[40] F. Panse, M. van Keulen, A. De Keijzer, and N. Ritter. Duplicate detection in probabilistic data. *Proceedings - International Conference on Data Engineering*, pages 179–182, 2010.

[41] F. Panse, M. van Keulen, and N. Ritter. Indeterministic handling of uncertain decisions in deduplication. *Journal of Data and Information Quality*, 4(2), 2013.

[42] R. Peeters, A. Primpeli, B. Wichtlhuber, and C. Bizer. Using schema.org Annotations for Training and Maintaining Product Matchers. *ACM International Conference Proceeding Series*, Part F1625:195–204, 2020.

[43] K. Peffers, T. Tuunanen, and M. A. Rothenberger. A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45–77, 2007.

[44] A. Primpeli, R. Peeters, and C. Bizer. The WDC training dataset and gold standard for large-scale product matching. *The Web Conference 2019 - Companion of the World Wide Web Conference, WWW 2019*, pages 381–386, 2019.

[45] C. Re and D. Suciu. Management of Data with Uncertainties. *16th ACM Conference on Information and Knowledge Management*, pages 3–7, 2007.

[46] P. Ristoski, P. Petrovski, P. Mika, and H. Paulheim. A machine learning approach for product matching and categorization. *Semantic Web*, 9(5):707–728, 2018.

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

[48] P. Sen, A. Deshpande, and L. Getoor. PrDB: Managing and exploiting rich correlations in probabilistic databases. *The VLDB Journal*, 18(5):1065–1090, 2009.

[49] A. Souihli and P. Senellart. Optimizing approximations of DNF query lineage in probabilistic XML. *Proceedings - International Conference on Data Engineering*, pages 721–732, 2013.

[50] Statista. Ranking of the most popular relational database management systems worldwide, as of January 2022. https://www.statista.com/statistics/1131568/worldwide-popularity-ranking-relational-database-management-systems/, 2022. Accessed 06 May 2022.

[51] M. Stonebraker and L. A. Rowe. The Design of POSTGRES. *ACM SIGMOD Record*, 15(2):340–355, 1986.

[52] The PostgreSQL Global Development Group.

PostgreSQL. https://www.postgresql.org/, 1996. Accessed 06 May 2022.

[53] The PostgreSQL Global Development Group. PostgreSQL 14.2 Documentation, 2022.

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

[55] M. van Keulen. Probabilistic Data Integration. *Encyclopedia of Big Data Technologies*, pages 1308–1315, 2019.

[56] M. van Keulen and A. De Keijzer. Qualitative effects of knowledge rules and user feedback in probabilistic data integration. *VLDB Journal*, 18(5):1191–1217, 2009.

[57] M. van Keulen and J. Flokstra. DuBio. https://github.com/utwente-db/DuBio, 2019. Accessed 03 May 2022.

[58] M. van Keulen, B. L. Kaminski, C. Matheja, and J. P. Katoen. *Rule-based conditioning of probabilistic data*, volume 11142 LNAI. Springer International Publishing, 2018.

[59] K. van Rijn. A Binary Decision Diagram based approach on improving Probabilistic Databases. Bachelor's thesis, University of Twente, 2020.

[60] B. Wanders and M. van Keulen. Revisiting the formal foundation of Probabilistic Databases. *Proceedings of the 2015 Conference of the International Fuzzy Systems Association and the European Society for Fuzzy Logic and Technology*, 89, 2015.

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

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

[63] Web Data Commons. Training Dataset and Gold Standard for Large-Scale Product Matching. http://webdatacommons.org/largescaleproductcorpus/v2/, 2018. Accessed 17 May 2022.

[64] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. *2nd Biennial Conference on Innovative Data Systems Research*, pages 262–276, 2005.

[65] R. J. Wieringa. *Design science methodology*. 2014.

[66] M. Wilke and E. Rahm. Towards Multi-Modal Entity Resolution for Product Matching. *CEUR Workshop Proceedings*, 3075, 2021.

[67] Z. Zhang, C. Bizer, R. Peeters, and A. Primpeli. MWPD2020: Semantic web challenge on mining the web of html-embedded product data. *CEUR Workshop Proceedings*, 2720, 2020.

APPENDIX

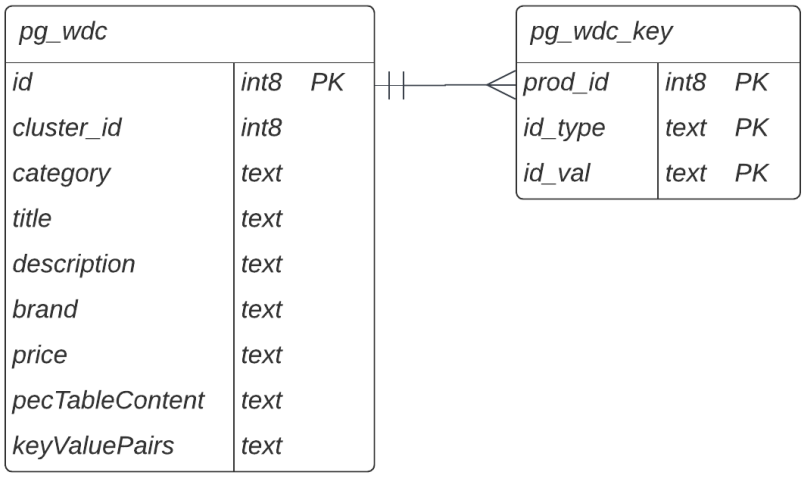
A. WDC PRODUCT OFFERS DATASET

This appendix gives an overview of the structure of the WDC product offer dataset.

A.1 JSON structure of a product offer


```
{
  "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 70 gb 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"
}
```


A.2 Database schema of the dataset



A.3 Dataset insight queries

```sql
-- (1) Gives the joined tables to have a full overvew.
SELECT *
FROM pg_wdc_small pws
LEFT JOIN pg_wdc_key_small pwks ON pws.id = pwks.prod_id
LIMIT 20;

-- (2) Create a view with all offers without NULL values.
CREATE VIEW nonull AS
SELECT *
FROM pg_wdc_small
WHERE category IS NOT NULL
AND title IS NOT NULL
AND description IS NOT NULL
AND brand IS NOT NULL
AND price IS NOT NULL
AND spectablecontent IS NOT NULL
AND keyvaluepairs IS NOT NULL;

-- (3) The amount of products without null values.
SELECT COUNT(*)
FROM nonull;

-- (4) Returns the percentage of offers that have no null values.
SELECT
       (SELECT CAST(COUNT(*) AS decimal) FROM nonull) / (SELECT COUNT(*) FROM
          ↪ pg_wdc_small) * 100;

-- (5) Returns the amount of product categories.
SELECT COUNT(DISTINCT category)
FROM pg_wdc_small;

-- (6) Returns the amount of products without a product categories.
SELECT COUNT(category)
FROM pg_wdc_small
WHERE category IS NULL;

-- (7) Returns the type of currencies and how often they occur.
SELECT NULLIF(currency, '') AS currency, COUNT(currency) AS amount
FROM (SELECT price, TRIM(regexp_replace(price, '[0-9]+', '', 'g')) currency
       FROM pg_wdc_small pws
       WHERE price IS NOT NULL) AS currencies
GROUP BY currency
ORDER BY amount DESC;

-- (8) Counts the amount of offers with >= 2 offer identifiers.
SELECT COUNT(*)
FROM (SELECT COUNT(*) AS ids
       FROM pg_wdc_small pws
       LEFT JOIN pg_wdc_key_small pwks ON pws.id = pwks.prod_id
       GROUP BY pwks.prod_id) AS count_id
WHERE count_id.ids >= 2;

-- (9) Returns the types of the product identifiers and how often they are used.
 SELECT DISTINCT id_type, count(id_type) AS products
 FROM pg_wdc_key_small
 GROUP BY id_type
 ORDER BY products DESC;

-- (10) Counts the amount of clusters that have >= 2 products in them.
SELECT COUNT(cluster_sizes.sizes)
FROM (SELECT COUNT(*) sizes
       FROM pg_wdc_small pws
       GROUP BY cluster_id) AS cluster_sizes
WHERE cluster_sizes.sizes >= 2;

-- (11) Returns for each cluster size the amount of clusters.
SELECT cluster_size, COUNT(cluster_sizes.cluster_size) AS amount
FROM (SELECT COUNT(pws.id) AS cluster_size
       FROM pg_wdc_small pws
       GROUP BY cluster_id) AS cluster_sizes
GROUP BY cluster_size
ORDER BY cluster_size ASC;
```

B. QUERIES FOR TESTING THE SETUP

This appendix provides the queries run with DuBio and MayBMS.

B.1 DuBio

B.1.1 Setting up the dataset

```
-- (1) Remove the cluster_id column from the dataset.
ALTER TABLE pg_wdc_small
DROP COLUMN cluster_id;

-- (2) Verify that the column is removed.
SELECT * FROM pg_wdc_small LIMIT 10;

-- (3) Create extensions to pgbdd of DuBio and pgcrypto for hasing.
CREATE EXTENSION pgbdd;
CREATE EXTENSION pgcrypto;

-- (4) Create table to store matching product pairs.
CREATE TABLE pg_wdc_match (
   cluster_id  BIGINT,
   prod_id     BIGINT,
   _sentence   Bdd,
   PRIMARY KEY (cluster_id, prod_id),
   FOREIGN KEY (prod_id)  REFERENCES pg_wdc_small(id)
);

-- (5) Create the main dictionary table.
CREATE TABLE _dict (name VARCHAR(20), dict DICTIONARY);

-- (6) Create the dictionary.
INSERT INTO _dict(name,dict) VALUES ('mydict', dictionary(''));

-- (7) Insert the random variables into the dictionary.
UPDATE _dict
SET dict=add(dict,'c1=1:0.7;c1=2:0.2;c1=3:0.1; c2=1:0.3;c2=2:0.3;c2=3:0.4; c3=1:0.3;
    ↪ c3=2:0.5;c3=3:0.2')
WHERE name='mydict';

-- (8) Verify that the dictionary is initialised and filled.
SELECT print(dict) from _dict WHERE name='mydict';

-- (9) Pseudorandomly fills the pg_wdc_match table with the products and their
    ↪ identified categories and likelihood.
INSERT INTO pg_wdc_match (cluster_id, prod_id, _sentence)
SELECT
   UNNEST(ARRAY[
      ('x' || ENCODE(DIGEST('a' || id::varchar(255), 'sha256'), 'hex'))::BIT(9)::int,
      CASE WHEN
         ('x' || ENCODE(DIGEST('b' || id::varchar(255), 'sha256'), 'hex'))::BIT(9)::
            ↪ int
         =
         ('x' || ENCODE(DIGEST('a' || id::varchar(255), 'sha256'), 'hex'))::BIT(9)::
            ↪ int
      THEN
         ('x' || ENCODE(DIGEST('b' || id::varchar(255), 'sha256'), 'hex'))::BIT(9)::
            ↪ int + 1
      ELSE
         ('x' || ENCODE(DIGEST('b' || id::varchar(255), 'sha256'), 'hex'))::BIT(9)::
            ↪ int
      END,
      -1]) AS cluster_id,
   id AS prod_id,
   UNNEST((
      SELECT ARRAY[
         Bdd(FORMAT('c%s=1', hash.val)),
         Bdd(FORMAT('c%s=2', hash.val)),
         Bdd(FORMAT('c%s=3', hash.val))]
      FROM (
         SELECT ('x' || ENCODE(DIGEST('c' || id::varchar(255), 'sha256'), 'hex'))::
            ↪ BIT(2)::int + 1 AS val
      ) AS hash
   )) AS _sentence
FROM pg_wdc_small;
```

B.1.2 Running queries on the dataset

```
-- (1) Count the amount of offers in cluster 200.
SELECT COUNT(*)
FROM pg_wdc_match
WHERE cluster_id = 200;

-- (2) Get an overview of all offers in cluster 200.
SELECT m.prod_id, m._sentence, p.*
FROM pg_wdc_small p
LEFT JOIN pg_wdc_match m ON m.prod_id = p.id
WHERE cluster_id = 200;

-- (3) Get the category and sentence from all product offers in cluster 200. Create a view from this.
CREATE VIEW cluster200 AS
   SELECT p.id, p.category, m._sentence
   FROM pg_wdc_small p
   LEFT JOIN pg_wdc_match m ON m.prod_id = p.id
   WHERE cluster_id = 200;

-- (4) Get a probability distribution for the category of cluster 200's product. Create a view from this.
CREATE VIEW likelihood_cluster200 AS
   SELECT c200.category, ROUND(AVG(prob(d.dict, c200._sentence)::NUMERIC),4) AS probability
   FROM cluster200 AS c200, _dict d
   WHERE d.name = 'mydict'
   GROUP BY c200.category
   ORDER BY probability DESC;

-- (5) Get categories with high enough confidence (>0.4) and display their current sentences. Create a view from this.
CREATE VIEW high_confidence_cluster200 AS
   SELECT array_agg(c200.id) AS id, c200.category, AGG_OR(c200._sentence) AS _sentence
   FROM cluster200 AS c200
   INNER JOIN (
          SELECT * FROM likelihood_cluster200
          WHERE probability > 0.4
   ) AS high_conf ON c200.category = high_conf.category
   GROUP BY c200.category;

-- (6) Insert random variables into the dictionary. These display conflicting attributes.
UPDATE _dict
SET dict = add(dict, 'd1=1:0.5;d1=2:0.3;d1=3:0.2;')
WHERE name = 'mydict';

-- (7) Insert new sentences
UPDATE pg_wdc_match
SET _sentence = (pg_wdc_match._sentence & Bdd('d1=1'))
FROM high_confidence_cluster200 AS hcc
WHERE pg_wdc_match.prod_id = ANY(hcc.id)
AND hcc.category = 'CDs_and_Vinyl';

UPDATE pg_wdc_match
SET _sentence = (pg_wdc_match._sentence & Bdd('d1=2'))
FROM high_confidence_cluster200 AS hcc
WHERE pg_wdc_match.prod_id = ANY(hcc.id)
AND hcc.category = 'Pet_Supplies';

UPDATE pg_wdc_match
SET _sentence = (pg_wdc_match._sentence & Bdd('d1=3'))
FROM high_confidence_cluster200 AS hcc
WHERE pg_wdc_match.prod_id = ANY(hcc.id)
AND hcc.category = 'Sports_and_Outdoors';

-- (8) Get the updated probability of the most likely categories.
SELECT c200.category, ROUND(AVG(prob(d.dict, c200._sentence)::NUMERIC),4) AS probability
FROM cluster200 AS c200, _dict d
WHERE d.name = 'mydict'
AND c200.category IN ('CDs_and_Vinyl', 'Pet_Supplies', 'Sports_and_Outdoors')
GROUP BY c200.category
ORDER BY probability DESC;
```

B.2 MayBMS

B.2.1 Setting up the dataset

```
-- (1) Remove cluster_id column from dataset.
ALTER TABLE pg_wdc_small
DROP COLUMN cluster_id;

-- (2) Verify that the column is removed.
SELECT * FROM pg_wdc_small LIMIT 10;

-- (3) Create table to store matching product pairs.
CREATE TABLE pg_wdc_match_setup (
   cluster_id  BIGINT,
   prod_id     BIGINT,
   probability FLOAT,
   PRIMARY KEY (cluster_id, prod_id),
   FOREIGN KEY (prod_id)  REFERENCES pg_wdc_small(id)
);

-- (4) Creating a table with probabilities for convenient inputting of these in the
    ↪ pg_wdc_match table
CREATE TABLE probs (probability float[4][3]);
INSERT INTO probs (probability) VALUES
   ('{{0.7,0.2,0.1},{0.3,0.3,0.4},{0.3,0.5,0.2},{0.5,0.1,0.4}}');

-- (5) Creating the UNNEST function available from PostgreSQL 9 onwards.
CREATE OR REPLACE FUNCTION unnest(anyarray)
RETURNS SETOF anyelement AS
   $BODY$
SELECT $1[i]
FROM   generate_series(array_lower($1,1), array_upper($1,1)) i
   $BODY$
LANGUAGE sql IMMUTABLE;

-- (6) Pseudorandomly fills the pg_wdc_match_setup table with the products and their
    ↪ identified categories and likelihood.
INSERT INTO pg_wdc_match_setup (cluster_id, prod_id, probability)
SELECT UNNEST(ARRAY[('x' || ENCODE(DIGEST('a' || id::varchar(255), 'sha256'), 'hex'))
    ↪ ::BIT(9)::int,
   CASE WHEN
      ('x' || ENCODE(DIGEST('b' || id::varchar(255), 'sha256'), 'hex'))::BIT(9)::int
      =
      ('x' || ENCODE(DIGEST('a' || id::varchar(255), 'sha256'), 'hex'))::BIT(9)::int
   THEN
      ('x' || ENCODE(DIGEST('b' || id::varchar(255), 'sha256'), 'hex'))::BIT(9)::int
          ↪ + 1
   ELSE
      ('x' || ENCODE(DIGEST('b' || id::varchar(255), 'sha256'), 'hex'))::BIT(9)::int
   END,
   -1]) AS cluster_id,
   id   AS prod_id,
   UNNEST((
      SELECT ARRAY[
         (SELECT probability[hash.val][1] FROM probs),
         (SELECT probability[hash.val][2] FROM probs),
         (SELECT probability[hash.val][3] FROM probs)]
      FROM (
         SELECT ('x' || ENCODE(DIGEST('c' || id::varchar(255), 'sha256'), 'hex'))::
             ↪ BIT(2)::int + 1 AS val
      ) AS hash
   )) AS probability
FROM pg_wdc_small;

-- (7) Create a probability space over the values in 'probability'. This makes the
    ↪ pg_wdc_match a probabilistic view.
CREATE VIEW pg_wdc_match AS
REPAIR KEY cluster_id IN pg_wdc_match_setup WEIGHT BY probability;

-- (8) Doing the same, but create a table this time.
CREATE TABLE pg_wdc_match_table AS
REPAIR KEY cluster_id IN pg_wdc_match_setup WEIGHT BY probability;

-- (9) Check the result. Three new coloumns are added: _v0, _d0 and _p0. When
    ↪ selecting from the view, these three columns are hidden.
SELECT *
FROM pg_wdc_match_table
ORDER BY cluster_id DESC
LIMIT 50;
```

B.2.2 Running queries on the dataset

```
-- (1) Count the amount of offers in cluster 200.
SELECT COUNT(*)
FROM pg_wdc_match
WHERE cluster_id = 200;

-- (2) Get an overview of all offers in cluster 200.
SELECT m.prod_id, m.probability, p.*
FROM pg_wdc_small p
LEFT JOIN pg_wdc_match m ON m.prod_id = p.id
WHERE cluster_id = 200;

-- (3) Get the category and sentence from all product offers in cluster 200. Create a
    ↪ view from this.
CREATE VIEW cluster200 AS
   SELECT p.id, p.category, m.probability
   FROM pg_wdc_small p
   LEFT JOIN pg_wdc_match m ON m.prod_id = p.id
   WHERE cluster_id = 200;

-- (4) Get a probability distribution for the category of cluster 200's product.
SELECT category, conf() AS prob
FROM (
   REPAIR KEY cluster_id IN (
      SELECT p.category, m.probability, m.cluster_id
      FROM pg_wdc_small p
      LEFT JOIN pg_wdc_match_setup m ON m.prod_id = p.id
      WHERE cluster_id = 200
   ) WEIGHT BY probability
) c200
GROUP BY category
ORDER BY prob DESC;

-- (4) Query 4 in the way it was supposed to work in the MayBMS tutorial.
SELECT c200.category, conf() AS prob
FROM cluster200 AS c200
GROUP BY c200.category
ORDER BY probability DESC;

-- (5) Get categories with high enough confidence (>0.066).
SELECT *
FROM (
   SELECT category, conf() AS prob
   FROM (
      REPAIR KEY cluster_id IN (
         SELECT p.category, m.probability, m.cluster_id
         FROM pg_wdc_small p
         LEFT JOIN pg_wdc_match_setup m ON m.prod_id = p.id
         WHERE cluster_id = 200
      ) WEIGHT BY probability
   ) c200
   GROUP BY category
) conf
WHERE prob > 0.066
ORDER BY prob DESC;

-- (6) Create another table to store new probabilities. These display conflicting
    ↪ attributes.
CREATE TABLE probs2 (prob float[4]);
INSERT INTO probs2 (prob) VALUES ('{0.4,0.3,0.2,0.1}');

-- (7) Update the setup table and insert the new probabilities.
ALTER TABLE pg_wdc_match_setup
ADD COLUMN probability2 FLOAT;

UPDATE pg_wdc_match_setup
SET probability2 = p.prob[1]
FROM probs2 p
WHERE pg_wdc_match_setup.prod_id = ANY(
   SELECT id
   FROM pg_wdc_small
   WHERE category = 'Computers_and_Accessories'
);

UPDATE pg_wdc_match_setup
SET probability2 = p.prob[2]
FROM probs2 p
WHERE pg_wdc_match_setup.prod_id = ANY(
```

```
76 |       SELECT id
77 |       FROM pg_wdc_small
78 |       WHERE category = 'Home_and_Garden'
79 |   );
80 |
81 |   UPDATE pg_wdc_match_setup
82 |   SET probability2 = p.prob[3]
83 |   FROM probs2 p
84 |   WHERE pg_wdc_match_setup.prod_id = ANY(
85 |       SELECT id
86 |       FROM pg_wdc_small
87 |       WHERE category = 'Office_Products'
88 |   );
89 |
90 |   UPDATE pg_wdc_match_setup
91 |   SET probability2 = p.prob[4]
92 |   FROM probs2 p
93 |   WHERE pg_wdc_match_setup.prod_id = ANY(
94 |       SELECT id
95 |       FROM pg_wdc_small
96 |       WHERE category = 'Tools_and_Home_Improvement'
97 |   );
98 |
99 |   -- (8) create a table with a combination of probabilities. The select query shows the
       ↪  structure of the table.
100 |  CREATE TABLE pg_wdc_match_updated AS
101 |      SELECT match1.*
102 |      FROM (REPAIR KEY cluster_id IN pg_wdc_match_setup WEIGHT BY probability) AS match1
           ↪ ,
103 |          (REPAIR KEY cluster_id IN pg_wdc_match_setup WEIGHT BY probability2) AS match2
104 |      WHERE match1.prod_id = match2.prod_id;
105 |
106 |  SELECT * FROM pg_wdc_match_updated LIMIT 10;
```

C. GANTT CHART FOR THE FINAL RESEARCH

Figure 1. Gantt chart of the planning