The evaluation of the conditioning task regarding probabilistic databases

JULIAN VAN DEN NIEUWENHUIZEN, University of Twente, The Netherlands

Probabilistic Databases aim to solve the problem of inconsistent or uncertain data collection and storage by using data representation on a probabilistic basis. By using this representation the database is able to store, retrieve and change uncertain data [2]. Probabilistic data Integration (PDI) is a type of data integration that achieves this goal. The PDI process contains 2 phases [2], the integration of data where data quality problems are not immediately solved but instead are represented as uncertainty in the probabilistic database. Afterwards the data will be continuously improved by gathering evidence through for example user feedback and improving the data accordingly. This research will focus primarily on the second step of the PDI process. The second phase of the PDI process is called conditioning. This paper shows results regarding the scalability of conditioning as well as the scalability of optimizing the database after a conditioning cycle. Furthermore, it shows in what situations optimizing the conditioned database has a positive effect.

Additional Key Words and Phrases: Probabilistic Database, PDI, conditioning, data quality, evidence, uncertain data

1 INTRODUCTION

Over the past decades, the introduction of probabilistic databases has become a viable substitution in fields like information retrieval, data cleaning, sensor data, tracking moving objects, crime fighting and computational science [1]. The ability to create, change, query and manipulate uncertain data is what sets a probabilistic database apart from a conventional database. In theory this should allow the user to retrieve this data in a much more efficient and reliable way. This research will make use of an extension on PostgreSQL named DuBio which adds probabilistic data types, functions and operators, making a database into a probabilistic database. The way to achieve an advantage over the conventional database involves the use of the second step of the PDI process. After doing a quick integration of the uncertain data, the database will in essence be based on the possible worlds theory [4]. Data can be represented in a number of different ways depending on the probability of the data. This creates a number of different possible worlds. Conditioning aims to introduce rules to eliminate a subset of possible worlds [3] [4] [5] . This is done by eliminating possible worlds that are not consistent with the newly proposed rule(s). The next step is to redistribute the eliminated probability mass over the possible worlds that do satisfy the newly proposed rule(s)[4]. We can make a distinction between hard and soft rules, hard rules completely eliminate possible worlds while soft rules redistribute the probability of the existing possible worlds [4]. More on this can be found in [4].

TScIT 41, July 5, 2024, Enschede, The Netherlands

Table 1. Example of database entry

name	eye_color	_sentence
Julian	blue	Bdd(X=1)
Julian	brown	Bdd(X=2)
Julian	green	Bdd(X=3)

Table 2. Example of dictionary

name	dict
mydict	X=1:0.5; X=2:0.3; X=3:0.2

2 BACKGROUND

2.1 Dubio

The probabilistic database system DuBio is being developed by the University of Twente since 2021. DuBio adds a number of additional functions to the existing capabilities of the SQL language. While SQL already has a wide range of functionality in creating, querying and altering databases, DuBio is aimed to specifically add functionality to accommodate the use of probabilistic concepts. To achieve this, several existing functions and capabilities of the SQL-language needed to be updated to include the creation of the required new data types and the ability to query and update the new data. The first step to making the use of probabilistic data possible, involves the creation of an efficient way to store the data. This required the creation of a new data type called "Bdd", this data type has the ability to store a large number of variables with their respective probability of being true. The Bdd data type has a numeric value between 0 and 1, based on the combined probability of all the variables and their individual probabilities. This newly created Bdd data is stored in the sentence column of a database entry. Table 1 shows a simple example of what this would look like. A person with three different possible data entries with a Bdd value stored in the _sentence column. In addition to this new data type and data entry column, a way to store variables with different possible values and their respective probabilities needed to be created. This is done with the use of a dictionary. Figure 2 shows a simple example of what this would look like. A dictionary with variable X, that can obtain different values with their respective probability.

2.2 Conditioning

Conditioning is part of the Probabilistic Data Integration process, or PDI. As mentioned in the introduction, the PDI process consists of 2 steps. The first step involves the quick integration of the probabilistic data into the database. The second step, called conditioning, is the process of improving the quality of the data. The quality improvement of the data starts with obtaining evidence of some form, this could be through data analysis, witness testimonies, user surveys and many more things. This evidence can be split into two categories, hard rules and soft rules. A hard rule implies that one of the values that a variable can attain, is either definitively true or false, meaning the probability of that value is either 1 or 0. A soft rule implies that the probability of the values that a variable can attain have to be redistributed. The process of applying a hard or soft rule to a probabilistic database involves updating the probabilities in the dictionary table to be in line with the new rule. Furthermore, if a hard rule is introduced, the sentence column in the database entries has to be updated as well.

3 IMPLEMENTATION

3.1 Conditioning

For the purpose of this research, an implementation of conditioning has been made. This implementation requires the introduction of a new rule, meaning the probabilities of the values of a variable will change. This implementation will make use of the introduction of a hard rule. The introduced hard rule will make the probability of one of the values of a variable one, meaning it is the only possible value for this variable. This implementation requires two steps:

- Updating the dictionary after the introduction of the hard rule
- Updating the sentence column of the database entries

The first step is accomplished by running an update statement on the dictionary. The probability of one of the values of a variable is set to 1, while the others are set to 0. The second step is accomplished by running an Update statement on the testdata table. The sentence column of the database entries will be appended with the only possible value for the updated variable, if the sentence does not contain this value yet. This will make all the database entries which previously did not contain the only possible value inherently false, since the probability of a database entry having two different values for the same variable is impossible. The exact update statements used for the implementation are listed in Appendix A.

3.2 Optimization

For the purpose of this research, an implementation of optimizing a conditioned database has been made. This implementation builds on the implementation of conditioning described in the previous subsection. The optimization of a conditioned database, is the process of deleting all database entries that have a probability of 0 after the conditioning cycle. This implementation requires two steps:

- Fetching all the probabilities of the database entries
- Deleting all database entries that have a probability of 0

The two steps are done simultaneously by using the testdata and dictionary table together in a delete statement. The statement will check if the probability of the database entry is equal to 0, if this is the case, the statement will then delete this database entry. The exact delete statement used for the implementation is listed in Appendix A.

4 RESEARCH QUESTIONS

The two main research question that this paper will try to answer are:

- How does the performance of the conditioning implementation scale with growing data?
- How does the performance of the optimization implementation scale with growing data?

The sub research questions to support the main research questions are:

- What kind of mathematical function best represents the relation between the data sample size and execution time of a conditioning cycle?
- What kind of mathematical function best represents the relation between the data sample size and execution time of optimizing a conditioned database?
- How does the performance of a conditioned database compare to the performance of a conditioned database which has been optimized?

5 METHODOLOGY

5.1 Preparing the database

5.1.1 Database structure. In order to answer the research questions, a suitable database needs to be setup. This process starts with defining the database structure. The database structure needs to have a unique id so that multiple database entries can be linked to this id. Furthermore, the database structure needs 2 variables that can have different values, for the purpose of the second part of this research. This also means that there needs to be a sentence to store these variables. And lastly the database structure needs to include a dictionary, that can store the different variables and their respective probabilities. The proposed database structure will be set up using phpPgAdmin.

5.1.2 Generating and inserting data. For the purpose of this research, the choice was made to use synthetically generated data. Because this is the first research being done on this subject, the obvious starting point is to use synthetic data. Real data would take many additional steps to prepare for testing which could take a considerable amount of time. The generation of the data is done through the use of a script. A single instance of data contains the following:

- An id, [1,2,3....]
- An eye color, [blue, brown, green]
- A hair color, [blonde, brown, black]
- A sentence, containing two variables with their respective value

There will be 9 different database entries for each unique id, since there are three different eye and hair colors. Table 3. gives a visual representation of the different database entries of one unique id. The script will also fill the dictionary with two variables X and Y, referring to eye and hair color respectively. The two variables will have three different values, referring to the different eye and hair colors. Table 4. gives a visual representation of the dictionary.

Table 3. Database entry

id	eye_color	hair_color	_sentence
1	blue	blond	Bdd(X=1 & Y=1)
1	blue	brown	Bdd(X=1 & Y=2)
1	brown	black	Bdd(X=1 & Y=3)
1	brown	blond	Bdd(X=2 & Y=1)
1	brown	brown	Bdd(X=2 & Y=2)
1	brown	black	Bdd(X=2 & Y=3)
1	green	blond	Bdd(X=3 & Y=1)
1	green	brown	Bdd(X=3 & Y=2)
1	green	black	Bdd(X=3 & Y=3)

Table 4. Dictionary

name	dict
mydict	X=1:0.5; X=2:0.3; X=3:0.2
	Y=1:0.5; Y=2:0.3; Y=3:0.2

5.2 Experiments

5.2.1 Conditioning. To test the scalability of conditioning a database, the performance of the conditioning will have to be measured at various data sample sizes. This includes the measuring of the times it takes to update the sentence column, as well as the time it takes to update the dictionary. With the measurements gathered, a graph can be constructed to study the relation between the performance and the data sample size.

5.2.2 Optimization. To test the scalability of optimizing a conditioned database, the performance of the optimization of a conditioned database will have to be measured at various sample sizes. With the measurements gathered, a graph can be constructed to study the relation between the performance and the data sample size.

5.2.3 Performance gain of optimizing a conditioned database. To test the performance gain of optimizing a conditioned database, a set of experiments will have to be conducted. A distinction is made between the retrieval of normal data and probabilistic data. For both methods of retrieval, the performance of a conditioned database will have to be compared to the performance of an optimized conditioned database. The statement to measure the performance of a conditioned database will have an addition that guarantees that the list of database entries retrieved, only contains database entries with a probability higher than 0. The statement needed to measure the performance of an optimized conditioned database will not require the addition, since the data left after optimizing is guaranteed to have a probability higher than 0. Furthermore, the performance of retrieving data will also be tested for probabilistic data. The key difference is the fact that these statements will have an additional column attached to all the retrieved database entries, containing the probability. This probability will be calculated with help of the dictionary table. The exact queries that will be used, can be found in Appendix A.

5.3 Measuring performance

5.3.1 Explain analyze. The PostgreSQL language has a wide range of functionality to measure performance. For the purposes of this research, the SQL command EXPLAIN ANALYZE will be used. By adding this command before any select/update/create statement, SQL will supply the user with a range of information. Most importantly it will supply the user with the execution time of the statement based on the plan it constructs. However, the statement following the explain analyze will be executed as well. For some of the statements that will be used for this research, we do not want that to happen. To prevent the execution of the statement, a script ensures that the SQL statements are not committed.

5.3.2 Method of measurement. It is important to choose a method of measurement that will ensure that the data gathered is of good quality. To ensure that the variation in execution times of different statements is minimized in the results, each statement will be run ten times in succession. The first measurement will be discarded because it is always somewhat to considerably slower than the subsequent measurements. The remaining measurements will be used to calculate an average execution time, which can be used to draw graphs and draw conclusions. All execution time measurements will be in milliseconds. To measure the scalability, the measurements will be done on a range of different sample sizes. The sample sizes are logarithmic with base 10, up to 100000. This will grant the ability to study the mathematical function that the measurements of scalability will present. The data samples have a logarithmic size of unique id's, meaning that the actual database sample size is nine times higher.

6 RESULTS

6.1 Conditioning

The measurements of a conditioning cycle include the execution times of updating the dictionary and updating the sentence column. This research is aimed at studying the scalability of the conditioning cycle. The execution time to update the dictionary can be disregarded for the purpose of this experiment, since the value will be the same regardless of the data sample size. The graph depicted in Figure 1. shows the different sample sizes with their respective execution times. Both axis have a logarithmic scale, so that a mathematical function can be seen. The line appears to be nearly straight, meaning that the execution times grow at the same rate as the data sample size. This points to a linear function.
The graph depicted in Figure 2. shows the average execution times of 10 unique id's per data sample size. The size of the bars, which represent the execution time, decrease at a steady rate, after which they increase slightly to a relatively consistent level. This points to a function that linearly decreases up until a certain point, after which it becomes a linear function. Further testing revealed the lowest relative execution time to be at approximately 5000 id's.

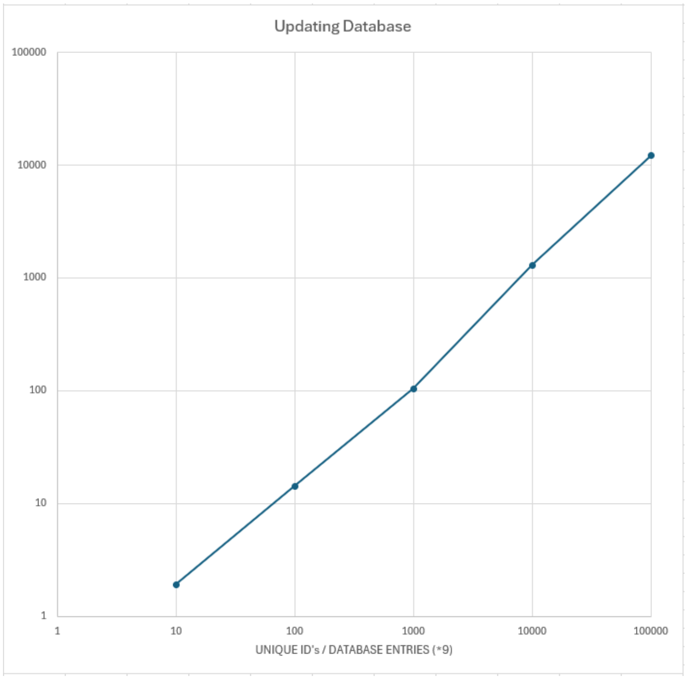

Fig. 1. Execution time for data sample sizes

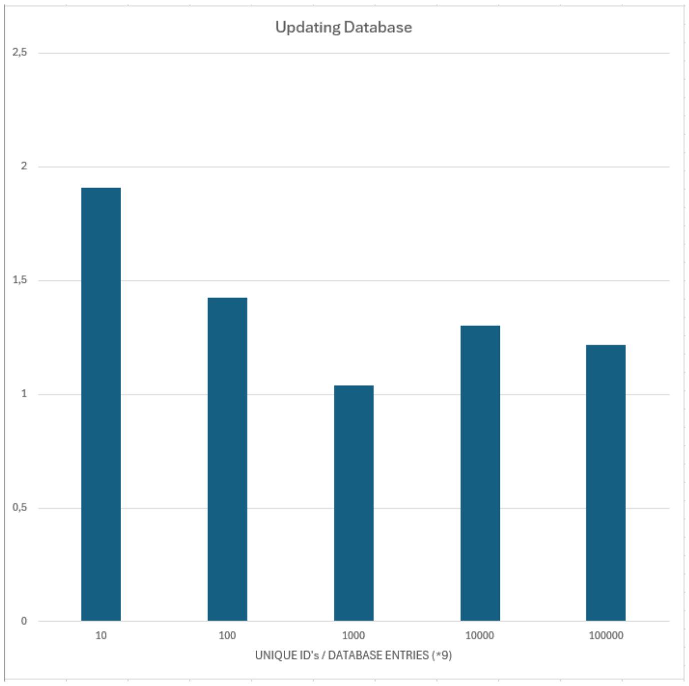

Fig. 2. Execution time per 10 IDS

6.2 Optimization

The measurements of optimizing the conditioned database include the execution times of optimizing the database by deleting the database entries that have a probability of 0. The conditioning cycle is responsible for making two thirds of the database entries suited for deletion. The graph depicted in Figure 3. shows the different sample sizes and their respective execution times. Both axis have a logarithmic scale, so that a mathematical function can be seen. With every increase in database sample size, it can be noted that the execution time increases at a slight exponential rate. Indicating that the optimization of a conditioned database will become more time consuming, the bigger the sample size gets.

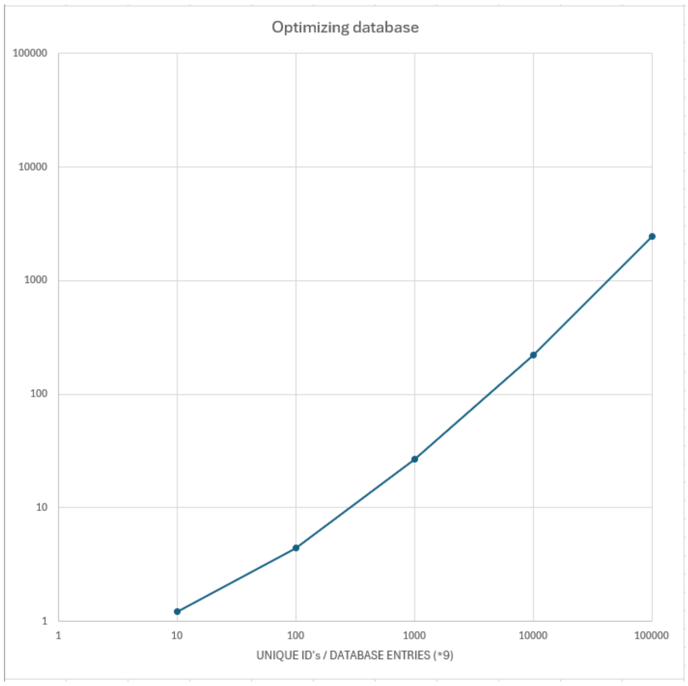

Fig. 3. Execution time for data sample sizes

6.3 Performance gain of optimizing a conditioned database

The goal of optimizing a conditioned database, is to decrease the execution times of all the subsequent queries and functions performed on the database. In the following sections, the performance gain of retrieving data through a select statement will be examined through the measurements presented. A distinction is made between the retrieval of normal data and the retrieval of probabilistic data. The measurements for the sample size of 100000 have been omitted since they made the graphs unreadable.

6.3.1 Normal data. The measurements of retrieving normal data from an optimized conditioned database and from a conditioned database show the execution time for various data sample sizes. The graph depicted in Figure 4. shows the comparison between execution times and data sample sizes. There are two lines in the graph, one represents the execution times of data retrieval from a conditioned database, the other represents the execution times of data retrieval from an optimized conditioned database. The lines show that the execution times of data retrieval without optimizing the database are faster at every data sample size. Indicating that optimizing the conditioned database, does not save time when retrieving data. However, the fact that optimizing the conditioned database only has to be done once, has to be considered. The graphs depicted in Figure 5. show the relation of execution times for subsequent select statements for different data sample sizes. The lines represent the conditioned database and the optimized conditioned database. The lines intersect somewhere between the first and second select statement. This points to the fact that after only 2 subsequent select statements, the optimized conditioned database has a faster execution time for every data sample size.

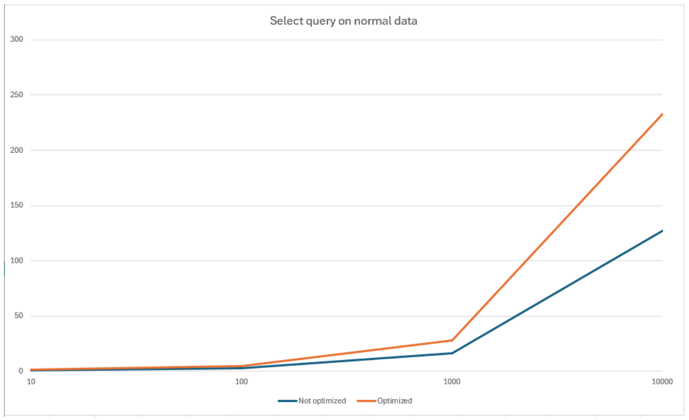

Fig. 4. Execution times of select statements

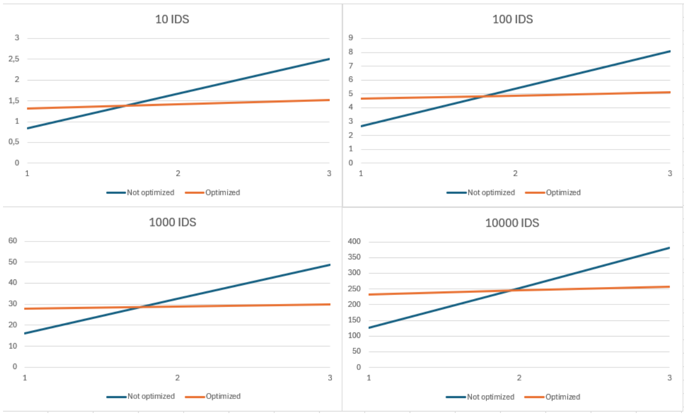

Fig. 5. Execution times of subsequent select statements

6.3.2 Probabilistic data retrieval. The measurements of retrieving probabilistic data from an optimized conditioned database and from a conditioned database show the execution time for various data sample sizes. The graph depicted in Figure 6. shows the comparison between the execution times and data sample sizes. The two lines represent the execution times of probabilistic data retrieval from a conditioned database, and from an optimized conditioned database. The lines indicate the same pattern as the retrieval of normal data. The retrieval on a conditioned database is faster in every case. However, again the fact that the optimization only has to be done once has to be considered. The graphs depicted in Figure 7. show the relation of execution times for subsequent select statements for different data sample sizes. The point where the lines intersect, indicates that the retrieval from the optimized conditioned database has surpassed that of the conditioned database. Furthermore, the point of intersection seems to decrease, the bigger the data sample size. Indicating that the performance gain increases, the bigger the data sample size becomes.

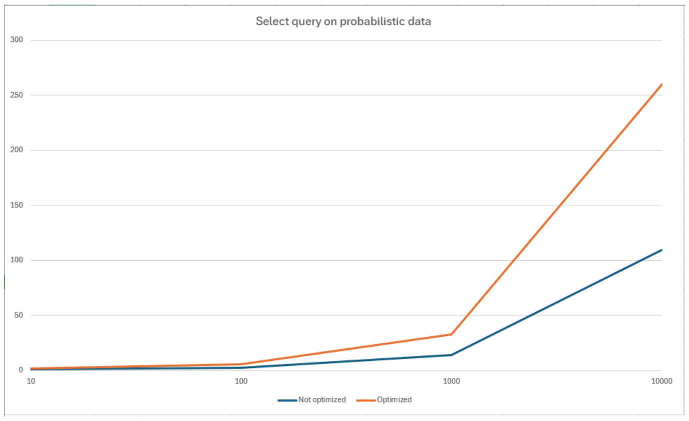

Fig. 6. Execution times of probabilistic select statements

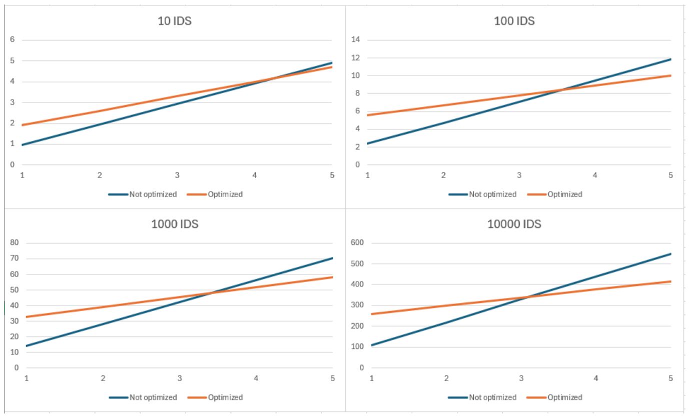

Fig. 7. Execution times of subsequent probabilistic select statements

7 CONCLUSIONS

The results on the scalability of the conditioning cycle indicate that the execution time scales approximately linear with the size of the database. The optimal performance was found to be around 5000 unique id's, which is around 45000 database entries. The results on the optimization of the database indicate two things. The scalability of the optimization is limited by sample size. However, the results indicate that the optimization does in fact provide a performance gain after only a few select statements. Moreover, the performance gain increases with database size when retrieving probabilistic data.

8 DISCUSSION

Based on the conclusion made regarding the performance gain of the optimization of the conditioned database, the advise for the creators of DuBio is to make an optimize function. The conclusion is based on the performance gain observed when using subsequent select statements, which is a real world use case. The optimize function would therefore actually save time and therefore is worth developing.

9 FUTURE WORK

The scope of this research is limited, therefore there is a lot of research still to be done regarding the evaluation of the conditioning task.

Firstly, this research is focused on the scalability of the conditioning task through varying data sample sizes. Further research can be done on the scalability of the complexity of the sentence. This would be a great addition to the overall research of the scalability.

Secondly, this research focused solely on the use of a select statement to test the optimization implementation. Further research can be done with a range of different statements.

REFERENCES

[1] Oktie Hassanzadeh and Renée J. Miller. 2009. Creating probabilistic databases from duplicated data. *The VLDB Journal* 18, 5 (01 Oct 2009), 1141–1166. https://doi.org/10.1007/s00778-009-0161-2
[2] Maurice Van Keulen. 2018. *Probabilistic Data Integration.* Springer International Publishing, Cham, 1–9. https://doi.org/10.1007/978-3-319-63962-8_18-1
[3] Christoph Koch and Dan Olteanu. 2008. Conditioning probabilistic databases. *arXiv preprint arXiv:0803.2212* (2008).
[4] Maurice van Keulen, Benjamin L. Kaminski, Christoph Matheja, and Joost-Pieter Katoen. 2018. Rule-Based Conditioning of Probabilistic Data. In *Scalable Uncertainty Management*, Davide Ciucci, Gabriella Pasi, and Barbara Vantaggi (Eds.). Springer International Publishing, Cham, 290–305.
[5] Hong Zhu, Caicai Zhang, Zhongsheng Cao, Ruiming Tang, and Mengyuan Yang. 2014. An Efficient Conditioning Method for Probabilistic Relational Databases. In *Web-Age Information Management*, Feifei Li, Guoliang Li, Seung-won Hwang, Bin Yao, and Zhenjie Zhang (Eds.). Springer International Publishing, Cham, 225–236.

A SQL AND DUBIO QUERIES

Goal	Query
Update the dictionary	UPDATE testdatabase._dict SET dict = upd(dict, 'X=1:1; X=2:0; X=3:0')
Update the sentence column	UPDATE testdatabase.testdata SET _sentence = _sentence & Bdd('X=1')
Query data with probability not 0	SELECT t FROM testdatabase.testdata t, testdatabase._dict d WHERE d.name = 'mydict' AND prob(d.dict, t._sentence) != 0
Query probabilistic data with probability not 0	SELECT t, prob(d.dict, t._sentence) FROM testdatabase.testdata t, testdatabase._dict d WHERE d.name = 'mydict' AND prob(d.dict, t._sentence) != 0
Query optimized data	SELECT * FROM testdatabase.testdata
Query probabilistic optimized data	SELECT t, prob(d.dict, t._sentence) FROM testdatabase.testdata t, testdatabase._dict d
Optimize the database	DELETE FROM testdatabase.testdata t USING testdatabase._dict d WHERE d.name = 'mydict' AND prob(d.dict, t._sentence) = 0

B RAW DATA

	10 UI 90 entries	100 UI 900 entries	1000 UI 9000 entries	10000 UI 90000 entries	100000 UI 900000 entries
Conditioning	2,536	15,354	171,84	1699,717	14238,866
	1,809	13,522	101,529	1333,757	13233,676
	2,205	12,564	103,731	1165,814	10603,249
	2,034	15,459	102,863	1286,068	10689,47
	1,685	14,091	101,401	1354,016	13920,596
	1,629	13,612	103,875	1194,332	13461,876
	1,953	16,264	104,553	1258,049	13642,271
	1,856	14,955	104,557	1420,608	10408,216
	1,735	13,39	105,003	1377,203	10908,769
	2,256	14,523	106,499	1317,405	12579,337
Optimizing Database	1,112	3,014	17,951	186,885	4822,191
	0,757	2,536	16,554	125,894	2449,012
	0,962	2,61	17,461	128,011	2558,104
	0,891	2,655	15,837	127,802	2412,541
	0,798	2,915	17,05	128,237	2401,413
	1,027	2,871	16,351	127,4	2395,215
	0,962	2,401	16,899	127,135	2438,297
	1,11	2,745	16,813	126,824	2517,303
	1,009	2,091	17,439	126,984	2590,416
	0,909	2,841	15,862	127,79	2426,581
Select Not Optimized	1,112	3,014	17,951	186,885	
	0,757	2,536	16,554	125,894	
	0,962	2,61	17,461	128,011	
	0,891	2,655	15,837	127,802	
	0,798	2,915	17,05	128,237	
	1,027	2,871	16,351	127,4	
	0,962	2,401	16,899	127,135	
	1,11	2,745	16,813	126,824	
	1,009	2,091	17,439	126,984	
	0,909	2,841	15,862	127,79	
Select Optimized	0,152	0,277	1,416	13,455	
	0,124	0,204	1,022	10,787	
	0,127	0,173	1,398	11,664	
	0,1	0,182	1,294	13,452	
	0,094	0,182	1,097	10,891	
	0,105	0,183	1,084	13,145	
	0,094	0,256	1,083	11,849	
	0,115	0,236	1,077	11,72	
	0,093	0,183	1,022	11,899	
	0,079	0,244	1,079	11,53	
Select Not Optimized PROB	1,121	2,728	15,629	172,178	
	1,099	2,197	14,318	109,449	
	1,104	2,52	14,6	130,547	
	1,087	2,231	15,088	109,884	
	0,877	2,543	15,11	109,247	
	0,969	2,557	14,095	108,616	
	1,011	2,359	15,234	108,271	
	0,895	2,718	15,291	109,671	
	0,775	2,571	14,857	108,763	
	1,003	2,535	13,912	109,913	
Select Optimized PROB	0,951	1,364	6,937	40,266	
	0,772	1,09	6,277	38,336	
	0,699	1,29	6,662	38,19	
	0,647	1,281	5,539	38,951	
	0,766	1,289	6,209	38,91	
	0,88	1,239	6,322	39,657	
	0,676	1,075	5,196	38,131	
	0,824	1,36	5,429	39,504	
	0,674	1,281	5,194	38,277	
	0,625	1,142	6,225	39,501	