Department of Informatics, University of Zürich

MSc Thesis

A dynamic website for a Temporal Probabilistic Database Implementation

Martin Leimer

Olten, SO, Switzerland

Matriculation number: 09-728-569

Email: martin.leimer@uzh.ch

August 18^{th}, 2014

supervised by Prof. Dr. M. Böhlen and K. Papaioannou

University of Zurich[UZH]

Department of Informatics

Acknowledgements

I would like to express my sincere gratitude to Aikaterini Papaioannou, who has supported me through the whole development of the application as well as the thesis. However, all this would not have been possible without the consent of the head of the database technology group. Therefore, I would like to appreciate Prof. Dr. Michael Böhlen, without whom I would not have been able to develop such a tool in such an interesting subject. In addition, many thanks to Ramona Wyss and Andrin Betschart for carefully counter-checking this thesis. Furthermore, sincere thanks to all other people, who have supported and guided me in any way through this thesis.

Abstract

Temporal probabilistic databases are a field of interest in recent research. In this thesis, we introduce a straightforward and user-friendly web application, which allows performing queries in a temporal probabilistic extension of PostgreSQL. Moreover, we propose algorithms and techniques to develop interesting visualisation tools that allow the user to analyse and understand the results of the algebra operations performed. Tree-like structures illustrate how and from which base tuples a result tuple is derived from, while bubble charts describe the impact of each base tuple on a specific result tuple. Predefined datasets and queries are provided to accelerate acquaintance with this web application whereas interactive features make it suitable for all users, regardless of their level of experience.

Zusammenfassung

Temporal probabilistische Datenbanken sind grosse Interessensgebiete in der heutigen Forschung. In dieser Arbeit führen wir eine übersichtliche und benutzerfreundliche Web-Applikation ein, welche die Ausführung von Abfragen auf einer temporal probabilistischen Erweiterung von PostgreSQL ermöglicht. Im Übrigen schlagen wir Algorithmen und Techniken vor um interessante Visualisierungen zu entwickeln. Diese ermöglichen dem Benutzer die Resultate der ausgeführten algebraischen Operationen zu analysieren und zu verstehen. Baum ähnliche Strukturen illustrieren wie und von welchen Basis-Tupeln ein Resultat-Tupel abgeleitet wurde, währenddessen Blasendiagramme den Einfluss jedes Basis-Tupels auf ein bestimmtes Resultat-Tupel aufzeigen. Im Weiteren eignet sich die Applikation für jeden Benutzer, unabhängig seines Erfahrungsstandes. Dies dank vordefinierten Datensets und Abfragen, sowie weiteren interaktiven Funktionalitäten.

Contents

Introduction

In this work, we develop a web application with analytical features for temporal probabilistic databases. One of those features is a tree representation which serves the need to understand how and which tuples influence the result probability, as well as if there are any dependencies between the tuples. Secondly, we introduce an algorithm to compute how big a result confidence is influenced by the probability of each base tuple. In this sense, we also provide a calculation method to compute the new result confidence given a probability change in one of its base tuples. Eventually, we show all this analysis in a clearly arranged bubble chart.

In Section 1, we introduce temporal probabilistic databases and provide the necessary reduction rules for all basic relational algebra operators. In this section, we also introduce the concept of lineage, which contributes to the correct computation of the confidence of a result tuple. The actual result confidence computation given lineage will be handled in Section 2. On top of that, we introduce a cutting-edge algorithm, with which we can calculate the effect of the probability of each base tuple on the confidence of the analysed result tuple.

In Section 3, we explain the features of our application in detail. In this regard, we also provide instructions on how to use it.

In Section 4, we describe how our concept was implemented in a web-application. This includes a description of which web techniques and libraries we used, as well as an explanation of the main algorithms.

In Section 5, we evaluate our application, draw concrete conclusions and propose ideas for future work.

1 Temporal Probabilistic Databases

Temporal Probabilistic Databases (TPDBs) enhance standard databases with temporal probabilistic relations and operators. Focussing on the relations first, temporal probabilistic relations always feature a time interval T and a probabilistic attribute p in addition to any other kind of attributes the relation has. While the time interval specifies from when to when a tuple will be valid, the probabilistic attribute specifies how certain it is that the corresponding event will occur at any time point during the specified time interval.

In order to perform query operations and calculate result confidences, we use lineage (λ) according to Sarma et al. [3]. Lineage is a boolean expression, which keeps track of the base tuples from which a result tuple is derived from. A base tuple is a tuple which is not derived from any other tuple.

Initially, each base tuple has a lineage expression (λ) which corresponds to the unique key (k) of that tuple. However, as soon as query operation produces a tuple which is made of more than one tuple, e.g. in an aggregation or a join, the lineage of both input tuples is combined using boolean operators like $\wedge, \vee, \neg$. In this regard, the actual boolean operator is chosen according to the relational algebra operation performed.

Temporal probabilistic relations consists always of three different kind of attributes:

- **Temporal attributes (T)**
  The temporal attribute consists of two timestamps, e.g. [2014-09-01, 2014-09-30). It describes from when (including) to when (excluding) an event described by the tuple and relation will take place. Thus, we can indirectly compute the events' duration by subtracting both timestamps.
- **Probabilistic attribute (p)**
  The probabilistic attribute defines the certainty that the event described by the tuple and relation will actually take place. It can be anything between $[0, 1]$, where 1 means that the event will take place for sure. In contrast, 0 means that the event is cancelled for sure.
- **Non-temporal and non-probabilistic attributes**
  This includes all attributes that are neither temporal nor probabilistic. In fact, this attributes are those which we find in a relation of a standard database, e.g. the name of a person, its id and so on.

Figure 1.1 shows an example of three temporal probabilistic relations. The first relation A (Availability) describes with which likelihood a person is available in the given time interval. For example, Ann from the computer science department is likely to be available with a probability of 95% from August, 1^{st} to the end of August, 29^{th}.

Secondly, relation I (Interest) gathers information from previous analysis about how likely it is that a certain person is interested in a topic in a specific time interval in the future. For example Janine is likely to be interested in TPDBs in September, 2014, whereas Ann has only a minor interest in it.

A (Availability)

k	p_{id}	$firstname$	$department$	T	p	λ
A_1	1	Ann	Computer Science	[2014-09-01, 2014-09-30)	0.95	A_1
A_2	2	Janine	Computer Science	[2014-08-26, 2014-09-08)	0.40	A_2

I (Interests)

k	i_{id}	p_{id_fkey}	$topic$	T	p	λ
I_1	1	1	TPDB	[2014-09-01, 2014-10-01)	0.40	I_1
I_2	1	2	TPDB	[2014-09-01, 2014-10-01)	0.95	I_2

E (Events)

k	e_{id}	i_{id_fkey}	$speaker$	T	p	λ
E_1	1	1	Andrin	[2014-09-01, 2014-09-08)	0.90	I_1

Figure 1.1: Temporal probabilistic relations

Eventually, relation E (Events) lists all events which might be held during a specific time interval. Each event has a speaker as well as foreign key to the I (Interest) relation, which describes the topic of the event. For example Andrin is likely to hold an event about TPDBs in the first week of September, 2014.

1.1 Temporal operators

Temporal databases do not only impose a time interval attribute on their relations, yet they also define temporal operators in order to query temporal probabilistic relations. The approach we use is based on Dignös et al. [1], who introduced three temporal operators called Normalization, Alignment and Absorption. While Absorption is essential within temporal databases, up to date there is no definition how to use it within temporal probabilistic databases. This as tuples with coinciding non-temporal attributes and a time interval that is equal or a subset of the time interval of the non-temporal matching tuple, may still have different lineages. Thus, as there exists no definition of how to combine those lineages when Absorption is applied, if at all, we put up with the fact that we might have duplicates of this kind.

On the other hand, Normalization ($\mathcal{N}$) and Alignment (Φ) may still be applied accordingly in temporal probabilistic database. However, this depends on how the event is defined. Assuming that the event will take place at any time point during the whole time interval with the same probability, no changes need to be made when time intervals are split into sub time intervals. In contrast, if the time interval had specified between which time points the event is likely to occur, additional constraints would have to be made. This, given that the duration of the event remains constant, it might be more likely that an event will occur, the longer the time interval is. However, as we specify the event duration with the given time interval, no overhead need to be made.

Apart from that, we need to show that each tuple produced by such a temporal operation is not derived from more than one input tuple. Indeed, Normalization ($\mathcal{N}$) and Alignment (Φ) do only create a set of output tuples for each input tuple, but no output tuple is derived from more than one input tuple. This even holds when the temporal operation is made against a different relation, as the splitting of a time interval does not need to be taken into consideration under the assumptions we imply. Thus, we can simply set the lineage of each output tuple to the lineage of its input tuple. Moreover, Normalization ($\mathcal{N}$) and Alignment (Φ) may be applied accordingly as they would in temporal databases.

$\pi_{p_{id\_fkey},topic,T,\lambda}(\mathbf{I}\Phi_{true}\mathbf{A})$

p_{id_fkey}	$topic$	T	λ
1	TPDB	[2014-09-01, 2014-09-08)	I_1
1	TPDB	[2014-09-01, 2014-09-30)	I_1
1	TPDB	[2014-09-30, 2014-10-01)	I_1
2	TPDB	[2014-09-01, 2014-09-08)	I_2
2	TPDB	[2014-09-01, 2014-09-30)	I_2
2	TPDB	[2014-09-30, 2014-10-01)	I_2

Figure 1.2: Temporal prob. alignment

$\pi_{firstname,T,\lambda}(\mathcal{N}_{A.p_{id}=B.p_{id}}(\mathbf{A};\rho_B(\mathbf{A})))$

$firstname$	T	λ
Ann	[2014-09-01, 2014-09-08)	A_1
Ann	[2014-09-08, 2014-09-30)	A_1
Janine	[2014-08-26, 2014-09-01)	A_2
Janine	[2014-09-01, 2014-09-08)	A_2

Figure 1.3: Temporal prob. normalization

Example 1 *Figure 1.2 shows a temporal alignment of the relation I (Interests) against A (Availability) on no common attribute. As we will see in section 1.2, this might be a use case for a later Cartesian product between those two relations. The input tuple I_1 is split into three tuples. The first time interval arises from Janine, who has a time interval of [2014-08-26, 2014-09-08), thus, the overlapping time interval is [2014-09-01, 2014-09-08). The second time interval arises from Ann, who has a time interval of [2014-09-01, 2014-09-30), therefore, the overlapping time interval is [2014-09-01, 2014-09-30). Finally, the third sub time interval is the remaining time interval of I_1 not covered by any tuple in the relation A (Availability) and therefore [2014-09-030, 2014-10-01).*

Example 2 *Figure 1.3 shows a temporal normalization of the relation A (Availability) on itself. This could be a step towards the aggregation on some attribute of that relation for example. The time interval of tuple A_1 is split into [2014-09-01, 2014-09-08) because of tuple A_2, as this is the sub time interval during which both tuples hold. Then, the second tuple for A_1 is produced because of tuple A_1 itself.*

1.2 Reduction Rules

Given this findings we adapt the reduction rules [1] for temporal probabilistic databases. In detail, we specify for each relational algebra operator, how the lineage expressions of the input tuples must be transformed into a lineage for the output tuple and how the time intervals should be adapted. For this, we distinguish between join operators ($\times, \bowtie, \rtimes, \ltimes, \bowtie$), each set operator ($\cup, \cap, -$) as well as selections and projections (σ, π).

Let **r** and **s** be temporal probabilistic relations over schema R^{TP}, θ be a predicate, F be a set of aggregation functions over $r.A$, $B \subseteq A$ be a set of attributes and OR, AND and NOT

functions respectively aggregation functions that combine the given lineage expressions with the defined connective. Then the reduction rules define a temporal probabilistic algebra with sequenced semantics as shown in Figure 1.4. [2]

Operator			**Reduction**
Selection	$\sigma_\theta^{TP}(\mathbf{r})$	$=$	$\sigma_\theta(\mathbf{r})$
Projection	$\pi_B^{TP}(\mathbf{r})$	$=$	${}_{B,T}\vartheta_{OR(\lambda)}(\mathcal{N}_B^{TP}(\mathbf{r};\mathbf{r}))$
High Aggregation	${}_B\vartheta_F^{TP}(\mathbf{r})$	$=$	${}_{B,T}\vartheta_{F,AND(\lambda)}(\mathcal{N}_B^{TP}(\mathbf{r};\mathbf{r}))$
Difference	$\mathbf{r} -^{TP} \mathbf{s}$	$=$	${}_{r.A,r.T}\vartheta_{OR(AND(r.\lambda,NOT(s.\lambda)))}((\mathcal{N}_A^{TP}(\mathbf{r};\mathcal{N}_A^{TP}(\mathbf{r};\mathbf{s}))) \bowtie_{r.A=s.A\wedge r.T=s.T}(\mathcal{N}_A^{TP}(\mathbf{s};\mathcal{N}_A^{TP}(\mathbf{s};\mathbf{r}))))$
Union	$\mathbf{r} \cup^{TP} \mathbf{s}$	$=$	${}_{r.A,r.T}\vartheta_{OR(r.\lambda)}(\mathcal{N}_A^{TP}(\mathbf{r};\mathcal{N}_A^{TP}(\mathbf{r};\mathbf{s})) \cup \mathcal{N}_A^{TP}(\mathbf{s};\mathcal{N}_A^{TP}(\mathbf{s};\mathbf{r})))$
Intersection	$\mathbf{r} \cap^{TP} \mathbf{s}$	$=$	${}_{r.A,r.T}\vartheta_{OR(AND(r.\lambda,s.\lambda))}((\mathcal{N}_A^{TP}(\mathbf{r};\mathcal{N}_A^{TP}(\mathbf{r};\mathbf{s}))) \bowtie_{r.A=s.A\wedge r.T=s.T}(\mathcal{N}_A^{TP}(\mathbf{s};\mathcal{N}_A^{TP}(\mathbf{s};\mathbf{r}))))$
Cartesian Product	$\mathbf{r} \times^{TP} \mathbf{s}$	$=$	$\pi_{r.A,r.T,s.A,s.T,AND(r.\lambda,s.\lambda)}((\mathbf{r}\Phi_{true}^{TP}\mathbf{s}) \bowtie_{r.T=s.T}(\mathbf{s}\Phi_{true}^{TP}\mathbf{r}))$
Inner Join	$\mathbf{r} \bowtie_\theta^{TP} \mathbf{s}$	$=$	$\pi_{r.A,r.T,s.A,s.T,AND(r.\lambda,s.\lambda)}((\mathbf{r}\Phi_\theta^{TP}\mathbf{s}) \bowtie_{\theta\wedge r.T=s.T}(\mathbf{s}\Phi_\theta^{TP}\mathbf{r}))$
Left Outer Join	$\mathbf{r} ⟕_\theta^{TP} \mathbf{s}$	$=$	$\pi_{r.A,r.T,s.A,s.T,AND(r.\lambda,s.\lambda)}((\mathbf{r}\Phi_\theta^{TP}\mathbf{s}) ⟕_{\theta\wedge r.T=s.T}(\mathbf{s}\Phi_\theta^{TP}\mathbf{r}))$
Right Outer Join	$\mathbf{r} ⟖_\theta^{TP} \mathbf{s}$	$=$	$\pi_{r.A,r.T,s.A,s.T,AND(r.\lambda,s.\lambda)}((\mathbf{r}\Phi_\theta^{TP}\mathbf{s}) ⟖_{\theta\wedge r.T=s.T}(\mathbf{s}\Phi_\theta^{TP}\mathbf{r}))$
Full Outer Join	$\mathbf{r} ⟗_\theta^{TP} \mathbf{s}$	$=$	$\pi_{r.A,r.T,s.A,s.T,AND(r.\lambda,s.\lambda)}((\mathbf{r}\Phi_\theta^{TP}\mathbf{s}) ⟗_{\theta\wedge r.T=s.T}(\mathbf{s}\Phi_\theta^{TP}\mathbf{r}))$

Figure 1.4: Reduction Rules

1.2.1 Selection and Projection

For selection and projection, the lineage of the output tuple usually corresponds to the lineage of the input tuple, as no input tuple has any influence on any other output tuple. However, as relational algebra operators are defined as duplicate eliminating, a projection might result in duplicates. In this event, the lineages of the similar tuples are combined with $\vee$-operators into one output tuple, such that the remaining output tuples can be removed.

$\sigma^{TP}_{p_{id\_fkey}=1}(\mathbf{I})$

i_{id}	p_{id_fkey}	$topic$	T	λ
1	1	TPDB	[2014-09-01, 2014-10-01)	I_1

Figure 1.5: Selection

$\pi^{TP}_{Department}(\mathbf{A})$

$department$	T	λ
Computer Science	[2014-08-26, 2014-09-01)	A_2
Computer Science	[2014-09-01, 2014-09-08)	$A_1 \vee A_2$
Computer Science	[2014-09-08, 2014-09-30)	A_1

Figure 1.6: Projection

Example 3 *Figure 1.5 shows all interests of the user with the $p_{id\_fkey} = 1$. In contrast, Figure 1.6 shows a temporal probabilistic projection of the relation A (Availability) on its department. If the relational algebra operator would not be duplicate eliminating, there would be four result tuples. However, as the tuple {'Computer Science', [2014-09-01, 2014-09-08)} occurs twice, the lineage of both tuples are combined with the $\vee$-operator into one result tuple with only one lineage expression ($A_1 \vee A_2$).*

1.2.2 Join operators

For the join operators $(\times, \bowtie, ⟕, ⟖, ⟗)$ the lineage of an output tuple is defined as the $\wedge$-combination of the corresponding lineage of the input tuples.

$\mathbf{A} \overset{TP}{\bowtie}_{p_{id}=p_{id\_fkey}} \mathbf{I}$

p_{id}	$firstname$	$department$	T	i_{id}	p_{id_fkey}	$topic$	T	λ
1	Ann	Computer Science	[2014-09-01, 2014-09-30)	1	1	TPDB	[2014-09-01, 2014-09-30)	$A_1 \wedge I_1$
2	Janine	Computer Science	[2014-08-26, 2014-09-01)	-	-	-	-	A_2
2	Janine	Computer Science	[2014-09-01, 2014-09-08)	1	2	TPDB	[2014-09-01, 2014-09-30)	$A_2 \wedge I_2$

Figure 1.7: Left outer join

Example 4 *Figure 1.7 shows a temporal probabilistic left outer join of relation A (Availability) with relation I (Interests) on the person id. Firstly, a temporal probabilistic join involves the alignment of the two relations against each other on the common attribute - the person id. Thus, the initial time interval of tuple A_1, which is [2014-09-01, 2014-09-30] is aligned into a set of time intervals corresponding to {[2014-09-01, 2014-09-30)}, whereas the set for A_2 is {[2014-08-26, 2014-09-01), [2014-09-01, 2014-09-08)}. Given that, we make the alignment for the Interests tuples, which results in a set of time intervals corresponding to {[2014-09-01, 2014-09-30), [2014-09-30, 2014-10-01)} for I_1 and {[2014-09-01, 2014-09-08), [2014-09-08, 2014-10-01)} for I_2 respectively. Finally, a left outer join on the common attribute as well as the time interval is done, which results as shown in Figure 1.7.*

1.2.3 Set operators

For the set operators $\cap, \cup, -$, lineage is handled independently for each operator. However, they all have in common that once the operation is executed, all duplicates are eliminated. In terms of lineage, we simply concatenate the lineage of each duplicate with the $\vee$-operator and set it as the lineage of the remaining output tuple, as we did for projection in section 1.2.1. But first, the set operation itself must be executed, which imposes the following lineage concatenation:

- **Union ($\cup$)**
  For the Union set operator, each output tuple corresponds to exactly one input tuple and all input tuples are present exactly once in the output. Thus, the lineage of the output tuple corresponds to the lineage of the input tuple.
- **Intersect ($\cap$)**
  For the Intersect set operator, each tuple of the first relation is matched against duplicates in the second relation. If there is no duplicate, the tuple is not present in the output. If there is a duplicate, the lineage of the tuple of the first relation is concatenated with all corresponding duplicates from the second relation using the $\wedge$-operator. All other values of the tuple remain as is.
- **Difference ($-$)**
  For the Difference set operator, each tuple of the first relation is matched against duplicates in the second relation. If there is no duplicate, the tuple is taken with the given lineage into the output of the operation. However, if there is at least one duplicate, the lineage of the corresponding duplicate is negated. Then, the lineage of the tuple of the first relation is concatenated with the negated lineage of the corresponding duplicates using the $\wedge$-operator.

2 Confidence and Effect Computation

The confidence of a result tuple gives us an indication of the probability with which the resulting event is going to happen. The effect of a base tuple on the investigated result tuple describes with which intensity and within which range the result confidence can be altered, by changing the probability of the corresponding base tuple.

Given lineage, we can clearly identify from which base tuples a result tuple is derived from. Furthermore, given a probability for each base tuple, we have certainly enough to compute the result confidence by evaluating lineage.

One approach to achieve this is by identifying all base tuples that occur in the lineage expression. Then, by creating a truth table using those base tuples, we are able to evaluate the set of Boolean values for which lineage evaluates to true. Given this set, we can finally compute the result confidence and hence, the effect each base tuple has on it.

By applying this process for each result tuples recursively, we eventually end up a result confidence for each result tuple.

A (Availability)

k	p_{id}	$firstname$	$department$	T	p	λ
A_1	1	Ann	Computer Science	[2014-09-01, 2014-09-30)	0.95	A_1
A_2	2	Janine	Computer Science	[2014-08-26, 2014-09-08)	0.40	A_2

I (Interests)

k	i_{id}	p_{id_fkey}	$topic$	T	p	λ
I_1	1	1	TPDB	[2014-09-01, 2014-10-01)	0.40	I_1
I_2	1	2	TPDB	[2014-09-01, 2014-10-01)	0.95	I_2

E (Events)

k	e_{id}	i_{id_fkey}	$speaker$	T	p	λ
E_1	1	1	Andrin	[2014-09-01, 2014-09-08)	0.90	I_1

Figure 2.1: Temporal probabilistic relations

In the following subsections, we are going to explain those steps in detail reusing our familiar example relations from section 1 as shown in Figure 2.1. Moreover, in order to compute any result confidences, we impose the following query:

$$\pi_{firstname,topic,speaker,E.T}(\mathbf{A} \bowtie^{TP}_{p_{id}=p_{id\_fkey}} (\mathbf{I} \bowtie^{TP}_{i_{id}=i_{id\_fkey}} \mathbf{E}))$$

In natural language, this query corresponds to the likelihood that a person will attend an event. This itself is depending on the probability that a person is available during the time interval the event takes place, that she or he is interested in the topic of the event, as well as that the event

actually takes place. Thus, the query is a temporal probabilistic join of the three relations, whose non-evaluated result can be seen in Figure 2.2.

$\pi_{firstname,topic,speaker,E.T}(\mathbf{A} \bowtie^{TP}_{p_{id}=p_{id\_fkey}} (\mathbf{I} \bowtie^{TP}_{i_{id}=i_{id\_fkey}} \mathbf{E}))$

$firstname$	$topic$	$speaker$	T	λ
Ann	TPDB	Andrin	[2014-09-01, 2014-09-08)	$A_1 \wedge I_1 \wedge E_1$
Janine	TPDB	Andrin	[2014-09-01, 2014-09-08)	$A_2 \wedge I_2 \wedge E_1$

Figure 2.2: Example query

Example 5 *Figure 2.2 describes that Ann and Janine may attend the event about TPDBs held by Andrin from September,* 1^{st} *up to the end of September,* 6^{th}.

2.1 Reduced truth table of a boolean expression

In order to compute the result confidence for a lineage expression, one approach is to create a truth table. For this, we first need to identify all base tuples of the given lineage expression. Then, by assigning to each base tuple a Boolean variable, we are able to create a truth table by considering all possible permutations of the Boolean values the Boolean variables can have.

Mathematically speaking, given n distinct base tuples, we can create 2^n combinations, as each Boolean variable can be either true or false. Therefore, we define the truth table as $T[1..2^n][1..(n+1)]$.

Example 6 *Given lineage* $A_1 \wedge I_1 \wedge E_1$ *from the first row in Figure 2.2, the truth table is as shown in Figure 2.3. It consists of exactly* $8 = 2^3$ *rows, as the lineage has exactly three different base tuples. Moreover, since the lineage consists of* $\wedge$*-operators only, it should be obvious that a row of a truth table only evaluates to true, if and only if all Boolean values of that row are set to true.*

Analogously, the process would be repeated for any other lineage. On that account, the truth table for lineage $A_2 \wedge I_2 \wedge E_1$ *would be symmetric.*

In order to enhance understanding how to compute final result confidence, we propose the creation of a reduced truth table. In detail, we drop all rows of the truth table that we do not need any more. Consequently, the reduced truth table only contains the rows of the truth table which evaluated to true. Thus, the reduced truth table can be defined as $R[1..m][1..(n+1)]$, where m represents the number of permutations that evaluated to true while n is the number of distinct base tuples.

A_1	I_1	E_1	$A_1 \wedge I_1 \wedge E_1$
True	True	True	True
True	True	False	False
True	False	True	False
True	False	False	False
False	True	True	False
False	True	False	False
False	False	True	False
False	False	False	False

Figure 2.3: Truth table

A_1	I_1	E_1	$A_1 \wedge I_1 \wedge E_1$
True	True	True	True

Figure 2.4: Reduced truth table

Example 7 *Figure 2.4 shows the reduced truth table derived from the truth table being represented in Figure 2.3.*

2.2 Confidence computation

Reconsider the reduced truth table $R[1..m][1..(n+1)]$, where m represents the number of permutations which evaluated to true, n the number of distinct base tuples and where r_{ij} represents the Boolean value stored in row i and column j. In addition, consider that b_j represents the base tuple of column j of the truth table T, while $b_j.p$ is its probability. Then, algorithm 1 computes the probabilities for the probability table $P[1..m][1..n]$, where p_{ij} represents the probability of row i and column j.

Algorithm 1 Computing the probability table P based on the reduced truth table R

```
1: for all i in 1 to m do
2:     for all j in 1 to n do
3:         if r_ij = true then
4:             p_ij ← b_j.p
5:         else
6:             p_ij ← 1 − b_j.p
7:         end if
8:     end for
9: end for
```

The probability table P is of the same size as the reduced truth table R, except for one column. Basically, each Boolean value of the reduced truth table R is replaced with the probability or the complement of the probability of the event described by the base tuple of the corresponding column.

A_1	I_1	E_1
0.95	0.40	0.90

Figure 2.5: Probability table for $A_1 \wedge I_1 \wedge E_1$

Example 8 *Figure 2.5 shows an example of the probability table after having applied algorithm 1.* p_{11} *equals to* 0.95*, because* r_{11} *equals to* $true$ *in Figure 2.4 and* $b_1.p = 0.95$*. Similarly,* p_{12} *equals to* 0.40*, because* $r_{11} = true$ *and* $b_1.p = 0.95$*.*

Example 9 *Consider the reduced truth* R *as shown in Figure 2.4. Moreover, assume that* $r_{13} = false$*. Then,* p_{13} *of Figure 2.5 would equal to* 0.10*, as we have to take the complement of the probability* $(1 - b_3.p)$ *of the event described by base tuple* b_3*.*

Having obtained the probability table P, we can compute the result confidence c. Firstly, for each row, we multiply the corresponding row-cells together. Then, by summing up over the m products, we are eventually left with the result confidence c for the given lineage expression, as shown in formula 1.

$$c(P) = \sum_{i=1}^{m} \left(\prod_{j=1}^{n} p_{ij} \right) \tag{1}$$

where p_{ij} represents the probability of row i and column j in the probability table.

A_1	I_1	E_1	Π
0.95	0.40	0.90	0.342
		Σ	0.342

Figure 2.6: Result confidence computation for $A_1 \wedge I_1 \wedge E_1$

Example 10 *Figure 2.6 shows an example of how to compute the result confidence given the probability table as shown in 2.5 and formula 1. Thus, Ann (*A_1*) is going to attend the event with a probability of 34.2%.*

2.3 Impact of the probability of a base tuple on a specific result tuple

In this section, we are going to describe how intense and within which range the probability of a result tuple can be changed, by changing the probability of an appearing base tuple.

In detail, we focus on a randomly selected base tuple being present in the corresponding lineage expression, from now on referred as the focused tuple. Then, by altering its probability and re-computing the resulting confidence, we are looking on what the lower and upper probability of the corresponding result tuple is. Given those two boundaries, we are able to compute how much the probability of a result tuple is influenced by the probability of the focused tuple. In

fact, the higher the difference between the lower and upper probability of the result tuple, the more is the result confidence influenced by the probability of the focused tuple. Finally, by doing this computation for all base tuples, we get the influence of each base tuple on the investigated result tuple.

In order to calculate the lower and upper bounds the result confidence can obtain, we set the probability of the focused tuple once to zero and once to one, the minimum and maximum values a probability can have. This is sufficient, as the result confidence is linear in the probability of a base tuple. Thus, the minimum and maximum value of a linear function are its boundaries.

Before going into detail how to compute those boundaries, we first proof that the result confidence computation is indeed linear in the manipulated probability of the focused tuple. In this regard, consider that $b_k, k \in \mathbb{N} | 1 \leq k \leq n$ is the focused tuple. Then, instead of taking the probability p_{ik} which is derived from $b_k.p$, we override $b_k.p$ with $x \in [0, 1]$ and thus use $f(x, r_{ik})$ instead of p_{ik}. On this account, $f(x, r_{ik})$ either equals the new probability x of the focused tuple or its complement, depending on whether the truth value r_{ik} equals $true$ or $false$.

Therefore the result confidence computation formula in dependency of the probability table P, the new probability x for the focused tuple b_k and the reduced truth table R is as shown in formula 2.

$$c(P, R, x, k) = \underbrace{\sum_{i=1}^{m} \Bigg(\underbrace{\underbrace{f(x, r_{ik})}_{\text{linear}} \cdot \underbrace{\prod_{\substack{j=1 \\ j \neq k}}^{n} p_{ij}}_{\text{constant}} \Bigg)}_{\text{linear}} = \underbrace{\sum_{i=1}^{m} \Bigg(f(x, r_{ik}) \cdot \prod_{\substack{j=1 \\ j \neq k}}^{n} p_{ij} \Bigg)}_{\text{linear}} \tag{2}$$

where $f(x, r_{ik}) = \begin{cases} x, & \text{if } r_{ik} = true \\ 1 - x, & \text{otherwise} \end{cases}$

Formula 2 shows that the result confidence computation is indeed linear in the new probability x with which we override the probability $b_k.p$ of the focused tuple. This as $f(x, r_{ik})$ is a linear function in x whether $r_{ik} = true$ or $r_{ik} = false$. Furthermore, the probabilities of all other base tuples p_{ij} where $i \neq k$ are fixed and thus constant. Therefore, the product of a linear function and constant variables is again linear. Similarly, the summation over a linear function is again linear and thus, the formula is linear in the overridden probability x of the focused tuple b_k.

Thus, given this findings, we can compute those two boundaries by reapplying the steps from section 2.2. Reconsider that we override $b_k.p$ with $x = 1$ for the first boundary and $x = 0$ for the second boundary. Then, the first and second boundary corresponds to the equations as shown in 3.

$$\text{first boundary} = \sum_{i=1}^{m} \left(f(1, r_{ik}) \cdot \prod_{\substack{j=1 \\ j \neq k}}^{n} p_{ij} \right) \tag{3a}$$

$$\text{second boundary} = \sum_{i=1}^{m} \left(f(0, r_{ik}) \cdot \prod_{\substack{j=1 \\ j \neq k}}^{n} p_{ij} \right) \tag{3b}$$

It remains to note, that the first boundary does not necessarily need to be the upper boundary. In fact, the formula for the result confidence is linear, but with increasing x, it can well be negative. This factor is completely depending on the lineage expression and thus the truth table. Unless the lineage expression contains no Boolean negations, it is not sufficient to say that the first boundary is the upper boundary.

A_1	I_1	E_1	eval
True	True	True	True

Figure 2.7: Reduced truth table for lineage $A_1 \wedge I_1 \wedge E_1$

A_1	I_1	E_1	Π
1.00	0.40	0.90	0.360
		Σ	0.360

Figure 2.8: 1^{st} boundary for lineage $A_1 \wedge I_1 \wedge E_1$ and focused tuple A_1

A_1	I_1	E_1	Π
0.00	0.40	0.90	0.000
		Σ	0.000

Figure 2.9: 2^{nd} boundary for lineage $A_1 \wedge I_1 \wedge E_1$ and focused tuple A_1

Example 11 *Figure 2.8 shows the first and Figure 2.9 the second boundary of the corresponding result confidence by adjusting the probability of the focused tuple A_1. They are based on the equation 3 as well as the reduced truth table for lineage $A_1 \wedge I_1 \wedge E_1$ as shown in Figure 2.7.*

In this case, the first boundary (0.360) actually corresponds to the upper boundary, while the second boundary (0.000) represents the lower boundary. Thus, we can increase the likelihood that Ann (A_1) will attend the event up to 36%, given that we can make sure that she will be available. On the other hand, if she will not be available, she surely cannot attend the event. Thus, the resulting probability is 0.000 which corresponds with the lower boundary we just calculated.

Example 12 *Consider that we created a truth table and probability table for the second lineage $A_2 \wedge I_2 \wedge E_1$ in Figure 2.2 as well. Moreover, consider that we computed the first and second boundary for this lineage expression, given that the availability of Janine (A_2) is the focused tuple. Then, the first and upper boundary is 0.855, while the second and lower boundary is 0.000.*

Given that this computation would reveal that the truth tables are almost identical and that their standard result confidence is equal, it might be surprising that their boundaries do not coincide at all. This fact can be explained that the multiplication of something big with an arbitrary chosen probability x is always higher than the multiplication with something small. Thus, knowing the two lineage expressions and that the probabilities of I_1, I_2 and E_1 are fixed, the upper boundary for lineage $A_2 \wedge I_2 \wedge E_1$ is higher, as Janine is much more interested in the topic than Ann ($I_2 > I_1$).

Example 13 *Consider lineage $a_1 \wedge \neg a_1$. Both, the upper and lower boundaries of the result confidence equate to zero, as no set of boolean values equates to $true$ in the truth table. Thus, we cannot increase the probability of this result tuple in any way.*

Given those two boundaries, it is left to show how intense a change in the probability of a base tuple is. This can be done by computing the marginal rate of the result confidence computation function, which is its first derivative. Thus, given formula 2, its first derivative is constant for any x.

In natural language, this implies that the absolute delta with which the result confidence changes is the same, whether we increase the overriden probability of the focused tuple from 20% to 30%, or 50% to 60%. Moreover, since the probability interval is restricted by the boundaries zero and one, whose delta is $1 - 0 = 1$, we can simply subtract the second boundary from the first boundary, which returns us the marginal rate of the focused tuple without the need of any division.

$$\text{marginal rate} = \text{first boundary} - \text{second boundary} \tag{4}$$

Depending whether the first boundary was actually the upper boundary, the marginal rate is increasing or not. This means that a higher probability of the focused tuple results in a higher probability of the result tuple if and only if the first boundary is the upper boundary. In contrast, if the first boundary is the lower boundary, the confidence of the result tuple decreases with a higher probability of the focused tuple.

The marginal rate itself has always to be understood as how much a change from 0% to 100% in the probability of the focused tuple would increase or decrease the result probability. However, by dividing it by 100, we are also able to compute how much an increase of 1% in the probability of the focused tuple has on the probability of the result tuple.

Example 14 *Given the upper and lower boundaries for the focused tuple* A_1 *and lineage* $A_1 \wedge I_1 \wedge E_1$*, as well as* A_2 *and lineage* $A_2 \wedge I_2 \wedge E_1$ *respectively, the marginal rate corresponds to* $0.360 - 0.000 = 0.360$ *and* $0.855 - 0.000 = 0.855$ *respectively. Thus, given that we want that people attend the event* E_1 *and that we can somehow change the availability of a person, it is best to give Janine (*A_2*) more free time as the marginal rate is higher for her than for Ann (*$0.855 > 0.360$*). This can be explained with the fact, that Janine is much more interested in the topic of the event. In contrast, even if Ann (*A_1*) had been available for sure, it is still unlikely (36%) that she would attend the event.*

It remains to note that the marginal rate as shown in equation 4 can never go beyond 100%, neither positive nor negative. If this had not been the case, we would end up with an invalid result probability being bigger than $1 = 100\%$ or lower than $0 = 0\%$. However, since both boundaries always lie between $0 = 0\%$ and $1 = 100\%$, their delta can never exceed $1 = 100\%$. Thus, the probability constraints are satisfied.

3 The application

Our implementation result is a website, which is accessible using the following hyperlink: `http://peter.ifi.uzh.ch/kat/`. However, due to server security properties, access is only permitted within or by VPN to the network of the University of Zurich.

The application itself features numerous utilities, which are not present in an application like pgAdmin. Thus, we will explain them in-depth in the following subsections. First, we show the benefit of a unique account, before going to explain each of the four tabs: *Executor, Datasets, Help* and *Account* as well as the visualisation of the result.

3.1 Independent workspaces

First of all, a user has to create a unique account in order to access the web application in full. After having signed up, each user is being presented with an independent workspace. Thus, only the user itself, or anyone who knows his login informations, has access to his own workspace. Therefore, the user can load and unload any relations and tuples according to his needs, without having to fear that another user will alter or delete them. In addition, the user could also store safety-critical data, as other default users cannot read them out.

Apart from that, we automatically load some predefined datasets into the users workspace, such that he can begin querying immediately. On this account, the user can either write his own query, or select a dataset and a relational algebra operator or a nested operation. Those feature will be explained in-depth in section 3.2 and 3.5.

For users being new to the application, we automatically launch a short tour, which describes the features of the application in detail by highlighting the different elements. It also simulates the selection of a predefined query as well as its execution. The tour explains the result table, as well as how result tuples can be further investigated. This includes explanations about the appearing graphs, which will be discussed in section 3.3 and 3.4 more precisely.

3.2 Executor

The executor features numerous functionalities, as shown in Figure 3.1. On the left hand side, a user has an overview over all relations that are currently loaded into his workspace. Moreover, by clicking on one of the relations, the user is being presented with their attributes and their type.

On the right hand side, the user can either write his own query or select one. For former, the user can write his query into the input area on the top right corner, before clicking on *Execute query*. Alternatively, he might choose a predefined dataset and a relational algebra operator or a nested operation with the two selectors shown in the middle. Given a user's selection, the corresponding PL/pgSQL-query including a natural language equivalent is being requested and loaded into the editable input area above. Thus the user does now not only understand what the query will do, but he is also being offered with the possibility to alter the query to his own needs, e.g. to other relations.

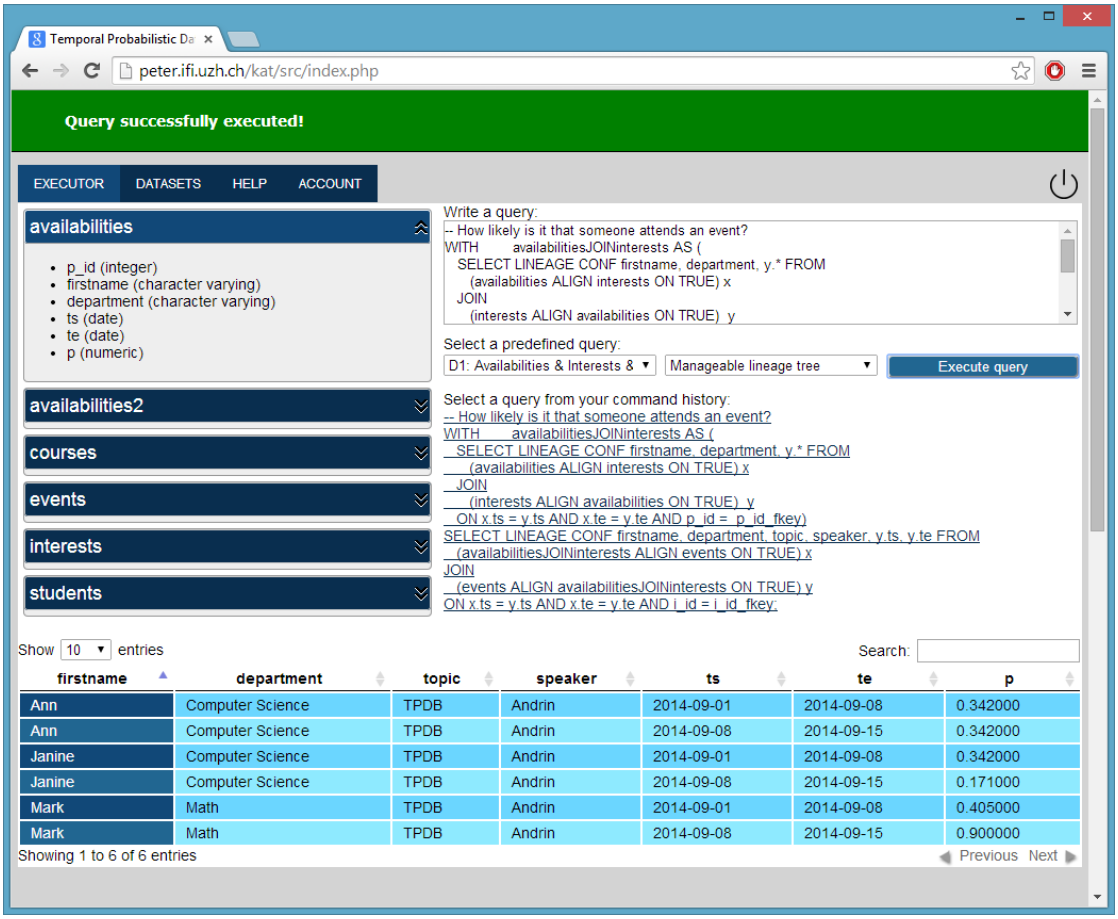

Figure 3.1: Screenshot of the executor

Apart from that, the user may select one of his 10 lastly executed queries in the list below. It serves the need of resuming the work after a break, or to revise an earlier result. By clicking on one of them, the query is re-executed instantly and copied into the editable input area for further alteration.

After having executed a query, a message will appear whether the query has executed successfully, or if there had been any errors. In the event that everything went smoothly and that the result is not empty, a result table will be shown like on the bottom of Figure 3.1. The result table itself is highly flexible and features some functionalities not being present in pgAdmin, a well-known standalone application to connect to PostgreSQL-servers and query them. The features are:

- **Full text search**
  On the top right corner of the table, there is a small search box. On typing any characters in it, all result tuples will be removed which do not contain the character sequenced in any of their attributes.

- **Sorting**
  The result is automatically sorted according the first column in a lexicographic increasing order. Moreover, by clicking on a column title, the sorting can be made for any other column too. In addition, by clicking on the same column again, the ordering of the tuples can be reversed. Thus, there is no need to write any ordering attributes in the query, unless required for selections.
- **Result tuple arrangement**
  Queries which result in a huge set of result tuples are often hard to understand, if all result tuples have to be investigated manually. It is likely that a result tuple is either investigated more than once or forgotten at all, especially if scrolling is indispensable. Therefore, we only list 10 result tuples per page, such that it is easier to go through all the result tuples. However, the user has the opportunity to see up to 100 result tuples per page, depending on his own preferences. This can be accomplished, by selecting the corresponding number of result tuples in the selector on the top left of the result table.
- **On click investigation**
  Given that the user has specified the keyword *LINEAGE* in his query, a click on a result tuple allows for further investigation of it in form of two visualisations as discussed in section 3.3 and 3.4.

3.3 Bubble chart

When the user clicks on a result tuple, a bubble chart is being computed. Its utility is that we can see how big the impact of a certain probability of a base tuple on the result confidence is and how they compare to each other. In fact, there is a bubble for each base tuple contributing to the corresponding result tuple having different colours and sizes as well as being differently allocated.

- **Allocation**
  The allocation of a bubble describes how significant it is for the result confidence computation. In fact, the higher the bubble is allocated on the screen, the higher is its impact. Thus, if we can somehow change the probability of the corresponding base tuple, we can dramatically change the confidence of the corresponding result tuple given that the bubble is high up in the graph.
- **Colour**
  There are three different colours defined, ranging from light blue to dark blue. Its meaning is whether the impact of a probability change in the base tuple is positive (light blue), neutral (blue) or negative (dark blue). In other words, if a bubble has a high allocation and is dark blue, an increase in the probability of the corresponding base tuple will result in a sharp decrease of the result confidence.
- **Size**
  The size of the bubbles tells us how high the probability of the given base tuple is. The bigger the bubble, the higher the probability. Thus, even if the bubble is light blue and

highly allocated, we cannot really increase the result confidence, as the corresponding has already a high probability. On the other hand, if it is our goal to minimize the result confidence, such a tuple should be our primary focus for changing its probability somehow.

Apart from that, by clicking on a bubble, we also list the attributes and values of the corresponding base tuple. This is essential, as each bubble is only labelled with an abstract identifier, containing the relation name and a result wide unique index for the tuple. Thus, given this feature, we exactly now, which tuple has which impact on the result.

Moreover, by hovering a bubble, the user is being presented with some key attributes, like the current probability of the result tuple, the marginal rate and within which range the result confidence could be altered by changing the probability of the investigated base tuple.

In general, we are interested in two kind of bubbles and their base tuples. Firstly, those which have a high impact and where we have a lot of room to change the result confidence. Secondly, those which have nearly no impact on the result. For the latter, we have to look for bubbles being allocated very low or having a blue colour independent of their size. For the former, we are only interested in the bubbles being high-up. However, we still have to distinguish four cases as shown in Figure 3.2.

Light blue	(+)	(-)
Dark blue	(-)	(+)
	Small (and high up)	**Big (and high up)**

Figure 3.2: Type of impact in a positive probability change in the base tuple

The (+) symbol in Figure 3.2 describes that we can significantly increase the result confidence by adjusting the probability of the base tuple. In contrast, the (-) symbol just describes the opposite, i.e. that we can significantly decrease the result confidence. Thus, depending, whether we are interested in increasing or decreasing the result probability, we do not only have to consider the size of a bubble, but also its colour. A light blue but small bubble gives us a lot of room for increasing the result probability, whereas a light blue but big small bubble only has room to decrease it.

Apart from that, any other bubble might still be interesting as well, though we have less room to increase or decrease the result confidence.

Example 15 *Consider that we want to know how likely it is that somebody attends an event. Under consideration of the reduction rules from section 1.2, we first have to make a join between the relations availabilities and interests on their common key. However, this requires that we temporarily align each relation with each other on the common key first. Having done the first join, we continue analogously for the second join with the relation events. The corresponding PL/pgSQL-query is as follows:*

```
-- How likely is it that someone attends an event?
WITH     availabilitiesJOINinterests AS (
    SELECT LINEAGE CONF firstname, department, y.* FROM
        (availabilities ALIGN interests ON p_id = p_id_fkey) x
    JOIN
        (interests ALIGN availabilities ON p_id = p_id_fkey)  y
    ON x.ts = y.ts AND x.te = y.te AND p_id = p_id_fkey)
SELECT LINEAGE CONF firstname, department, topic, speaker, y.ts, y.te FROM
    (availabilitiesJOINinterests ALIGN events ON i_id = i_id_fkey) x
JOIN
    (events ALIGN availabilitiesJOINinterests ON i_id = i_id_fkey) y
ON x.ts = y.ts AND x.te = y.te AND i_id = i_id_fkey;
```

Given that, an extract of the result is shown in Figure 3.3, where the result tuple {Janine, Computer Science, TPDB, Andrin, 2014-09-01, 2014-09-08, 0.342000} is being further investigated. In Figure 3.3, we can clearly identify three bubbles. Therefore, the result confidence computation of the investigated result tuple is dependent on three different base tuples: availabilities.1, events.1 and interests.1, where the name corresponds to the relation name and the id

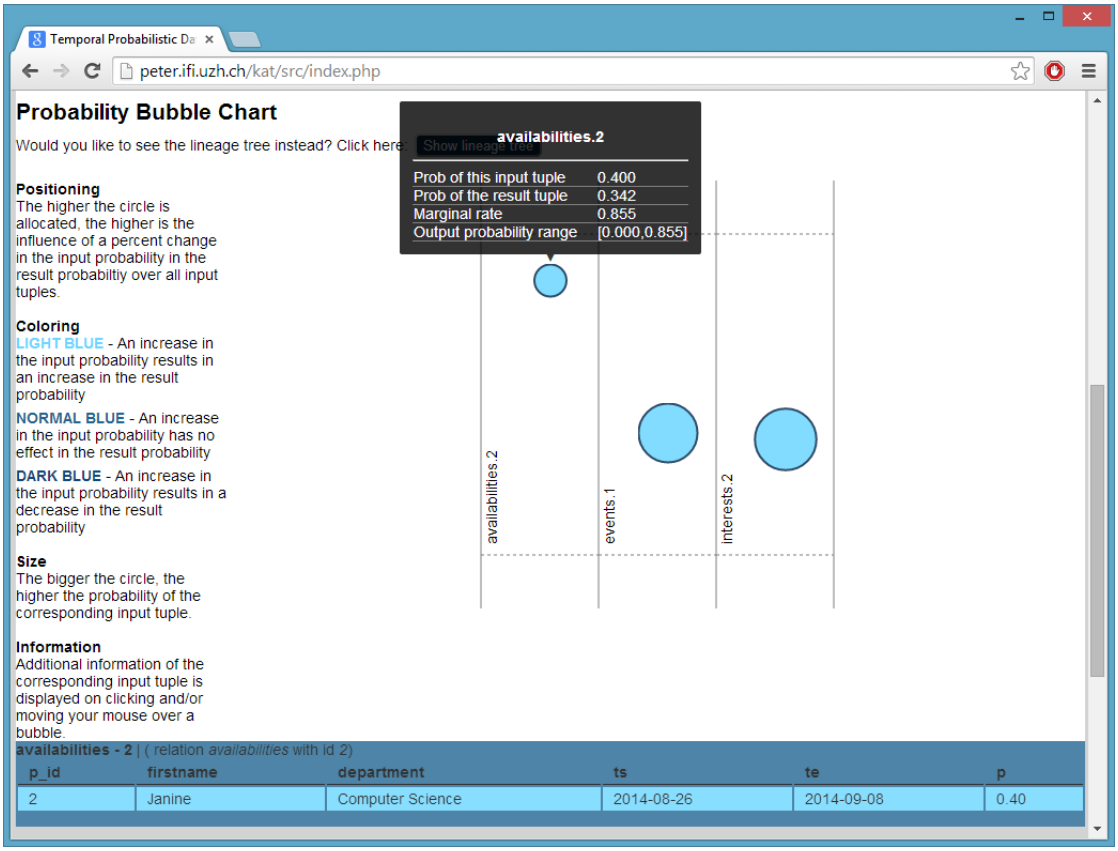

Figure 3.3: Query example: How likely is it that someone attends an event?

to an arbitrary chosen but result wide unique id.

Assuming that we are interested in having as many people possible attending an event, our view immediately falls to the first bubble {availabilities.2}, representing base tuple {2, Janine, Computer Science, 2014-08-26, 2014-09-08, 0.40}. The reason for this is that her bubble is high up, small and light blue. Therefore, if we can somehow increase the chance of her availability, currently being 40%, it significantly increases the result confidence up to a probability of 85.5%.

3.4 Lineage tree

The lineage tree is being presented on click on the corresponding button appearing together with the bubble chart. It shows how and from which base tuples the result tuple is derived from. Usually, each non-leaf node is the result of exactly one relational algebra operation, except for duplicate elimination and the difference operator. For duplicate eliminating operators like Union, Intersection or Difference, there might be also a $\vee$-node as a parent of multiple $\wedge$-nodes, supposing that there exists at least one duplicate. On the other hand, for Difference, also the $\neg$ nodes belong to their parent $\wedge$-nodes.

Given that a certain node has more than three children, we only show three of them at once. This makes it easier to understand and navigate through the tree. However, if someone wants to explore the hidden children, he simply has to click on the yellow triangle pointing up or down. A click on those triangles will result in the loading of the next three children in the selected direction. Once all children are traversed, the triangle disappears and only the triangle for the other direction remains visible.

Apart from that, the non-leaf nodes are expandable and collapsible by clicking on them. In addition, the tree can also be enlarged and scaled down by holding the mouse pointer onto the tree and turning the mouse wheel. Moreover, by clicking on a leaf node, the corresponding attributes and values of the base tuple will be shown. Furthermore, a message appears describing how many times this tuple is being present in the lineage tree. If any base tuple is more than once present, some subtrees are depended of each other, and thus some result confidence computation algorithms may no longer be adequate. In addition to showing a message, we also highlight and underline all those duplicates in the lineage tree as long as their corresponding parent nodes are fully expanded.

Example 16 *Figure 3.4 shows an example of the lineage tree of the investigated result tuple in Figure 3.3 with lineage $A_1 \wedge I_1 \wedge E_1$. By expanding all nodes, we can identify three leaf nodes and two $\wedge$-nodes. Reconsidering our query, we know that we have made two joins. Firstly, a join between availabilities and interests and then all together with events. By considering the lineage tree now, we can clearly identify that those two joins have been mapped into $\wedge$-operators in the same way, as we formulated the query.*

Given that, we can now look for any duplicates, which are obviously not present. Thus, algorithms that are more efficient could be used to compute the result confidence for this tuple. However, this is going beyond the scope of this thesis.

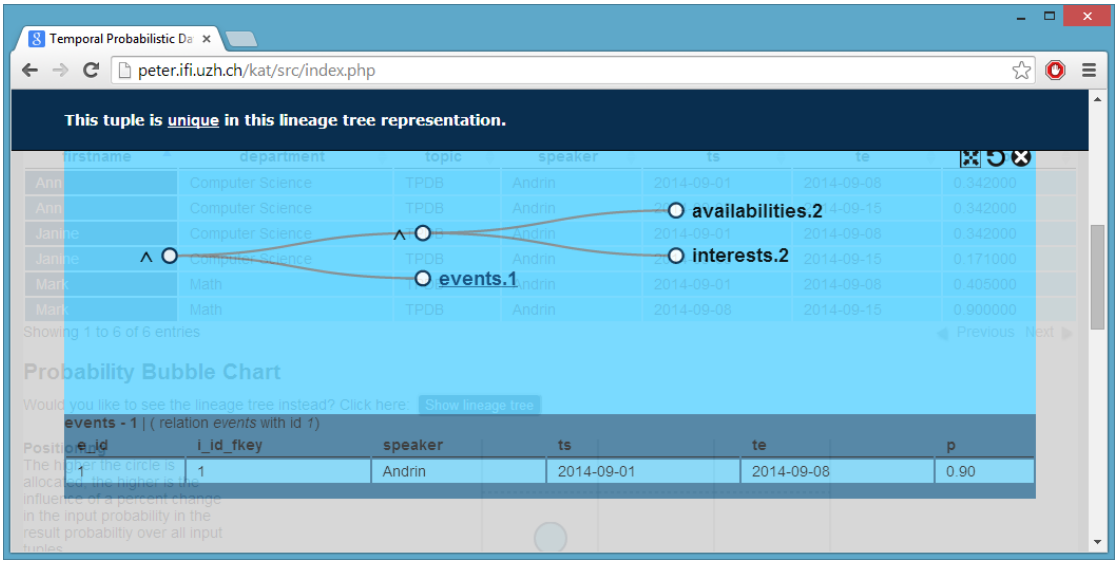

Figure 3.4: Lineage tree for the 2^{nd} result tuple of Figure 3.3

3.5 Datasets

The *Datasets* tab lists all datasets that can be loaded into the current workspace. It does not only allow renewing the dataset after having altered it, but also gives a description of the relations and tuples it contains.

On that account, we provide the following two temporal probabilistic datasets:

- **Availabilities & Interests & Events**
  This datasets consits of four relations called *Availabilities*, *Availabilities2*, *Interests* and *Events*:
  - *Availabilities*
    People, who are likely to be available during some specific time interval. Its attributes are:
    - *p_id* - the unique id of a person
    - *firstname* - the first name of a person
    - *department* - the department to whom someone belongs
    - *ts, te* - the time interval of the availability
    - *p* - the probability that somebody is available
  - *Availabilities2*
    A similar relation to *Availabilities*, but with less tuples
  - *Interests*
    Based on data from previous analysis, how likely is it that a person will be interested

in a certain topic during a specific time interval. Its attributes are:

  * *i_id* - the unique id of an interest
  * *p_id_fkey* - the person which is interested in this topic
  * *topic* - the topic of the interest
  * *ts, te* - the time interval when somebody might be interested
  * *p* - the probability that somebody is interested in this topic

- *Events*
  Events, which might be held on a certain topic during a specific time interval. Its attributes are:
  * *e_id* - the unique id of an event
  * *i_id_fkey* - the interest on which an event is hold
  * *speaker* - the speaker of the event
  * *ts, te* - the time interval when the event might take place
  * *p* - the probability that the event will take place

By joining all three relations together on their keys and foreign keys, we can answer questions like: "How likely is it that someone attends an event?" Another query could be which date we shall choose if at least one person of each department shall attend it, while we are flexible when to hold the event. Thus, this dataset is best for novices, as it is straightforward and allows querying interesting questions. Moreover, lineage might not necessarily be very complex.

- **Duplicate paradise**
  This datasets consist of two relations called *Students* and *Courses*:
  - *Students*
    For each student, we list all the courses he might attended in the future. Its attributes are:
    * *name* - the name of the student
    * *course* - the course name a student attends might attend
    * *ts, te* - the time interval when the student wants to take the course
    * *p* - the probability that the student books this course
  - *Courses*
    For each semester, we list all courses, which might be offered. Its attributes are:
    * *course* - the name of the course
    * *location* - the location where the course might take place
    * *ts, te* - the time interval when the course might take place

* p - the probability that the course will actually be held

As we list for each student all courses he might attend, we have a 1:N-relation. Therefore, if we project on the students names we are left with a huge amount of duplicates. This offers us the opportunity to execute queries that result in more complex lineage expressions and therefore bigger lineage trees. Moreover, one might also try to understand the difference between doing a projection followed by a duplicate elimination and a simple aggregation. While the aggregation will result that lineage expressions will be concatenated with $\wedge$-operators, there will be $\vee$-operators for projection followed by a duplicate elimination.

Apart from that, we also provide some sample queries for those datasets, which are selectable in the corresponding selector in the *Executor* tab. One can choose between a relational algebra operator and a nested operation. If a user chooses a relational algebra operator, a query will be chosen which maps the operator in PL/pgSQL and which is suitable for the selected dataset. Otherwise, we show a meaningful query with nested operations or a query which results in interesting visualisations.

The advantage of predefined queries is significant, not least because of the constraints of sequenced semantics as well as the temporal probabilistic PL/pgSQL-keywords like *LINEAGE, CONF, NORMALIZE* and *ALIGN*, which were introduced in the PostgreSQL-implementation we use.

Given those predefined queries, we dis-burden the user as much as possible, such that he can immediately focus on the results the query produces. In the event that the user has loaded his own dataset, he can easily adapt a predefined query to his own needs, allowing for efficient and less error-prone query writing.

Note: A temporal probabilistic query might be executable even if it is not correctly written. The possibly most common error is that a user does not specify all necessary time adjustments and thus violating the concept of sequenced semantics which we use. This sort of error cannot be detected automatically, as it might be done on purpose, e.g. if a user is sure, that no time adjustments are needed. Therefore, there will be a result in either case; however, it might be false and not necessarily recognizable as such.

3.6 Account and Help

In the *Account* tab it is possible to change the password without having to do a new registration or losing any data. Moreover, an account can be safely removed, by clicking the corresponding button in the same tab. This however has to be reconfirmed by entering the current password of that account, in order to prevent accidental deletion.

The *Help* tab lists a brief review about temporal probabilistic databases and its queries. It contains the reduction rules, which describe how to write temporal probabilistic queries as well as a button to restart the tour that was shown when the user first logged in to his account.

4 Implementation

The implementation is based on the following technologies:

- **HTML** (Hyper Text Markup Language)
  HTML defines the structure of a website by setting up elements. In addition, those elements may or may not contain any content like text for example.
- **CSS** (Cascading Style Sheets)
  CSS defines where these elements shall be placed on the screen and what they shall look like. In addition, definitions about whether elements shall be hidden or brought into the front can be made.
- **JavaScript** (JS)
  Javascript is a web language to alter elements and their content dynamically. Either by executing local functions, or by sending and retrieving data to and from a server, which is also known as XMLHttpRequest.
- **PHP**
  PHP is a server-side scripting language, meaning that unlike JavaScript, it is not executed on the clients computer, but on the server itself. Thus, it not only allows connections to other server interfaces as a PostgreSQL-Server for example, but can also hide and store algorithm details and credentials securely from the user.
- **PostgreSQL**
  PostgreSQL, also known as Postgres, is an object-relational database management system. We use it in altered version of Postges 9.2.4, which supports temporal probabilistic databases.
- **PL/pgSQL** (Procedural Language/PostgreSQL Structured Query Language)
  The queries are written in PL/pgSQL, a procedural language belonging to PostgreSQL.

Moreover, we make us of the following JavaScript libraries:

- **jQuery**
  Library to easier access and modify elements; "Write less, Do More"
- **DataTables**
  A jQuery Plugin to create flexible and interactive tables, e.g. the result table
- **Data Driven Documents**
  Library for creating dynamic charts, e.g. the bubble chart or the lineage tree representation
- **jMenu**
  A jQuery Plugin to generate beautiful navigation bars
- **Smallipop**
  A jQuery Plugin to create tours, e.g. the introduction tour when a new user signs up

In general, the web application we created is rather flexible on the PostgreSQL implementation we use, meaning that the PostgreSQL implementation is exchangeable. This as most of the

code written is independent, only the interface has its dependencies on the actual PostgreSQL implementation. Those are:

- **PL/pgSQL Synopsis**
  We require that the PL/pgSQL Synopsis is according to Figure 4.1.
- **Lineage**
  We require that lineage returns a boolean expression in form of a string without white-spaces. Moreover, each base tuple must be of the structure *tableoid.oid*. Furthermore, children of boolean operators must be enclosed within parentheses and siblings of a boolean operator must always represent the same boolean operator. Eventually, the $\neg$ operator must be represented with the character $-$, $\wedge$ with $*$ and $\vee$ with $+$.

  Example 17 *A valid string would be* $(30001.10001) + (30001.10002) + (30001.10003)$. *In contrast,* $30001.10001 + 30001.10002 \wedge (30001.10003)$ *would not satisfy this condition, as the children* 30001.10001 *and* 30001.10002 *of the* $\vee$*-operator are not enclosed within parentheses and as* $\wedge$ *is a different operator than* $\vee$ *but a sibling of him.*

If those requirements are not satisfied, the interface has to be altered correspondingly.

In section 4.1, we describe the PL/pgSQL synopsis we use more thoroughly. Then, we describe what needs to be considered when setting up a server in section 4.2, before we explain how we achieved implementing independent workspaces in section 4.3. Eventually, we describe two of the main algorithms of our web application in section 4.4 and 4.5. For further documentation, we refer to the source code directly.

4.1 PL/pgSQL Synopsis

An extract of the PL/pgSQL synopsis of the used PostgreSQL 9.2.4 implementations, corresponds to Figure 4.1.

- **LINEAGE**
  The keyword *LINEAGE* tracks from which base tuples the result tuple is derived from, which is essential to obtain the lineage tree and compute the bubble chart.
- **CONF**
  If the keyword *CONF* is specified, the confidence for all result tuples are computed. However, due to exponential complexity in the number of different base tuples from which the result tuple was derived from, we suggest not to use it. Instead, the result confidence of a result tuple can be compute more efficiently by clicking on a result tuple and looking up its value in the bubble chart.
- **NORMALIZE**
  The keyword *NORMALIZE* is used to execute a temporal normalization on the specified relations. For this, we firstly specify the relation or subquery which shall be normalized against another one. Then, secondly the keyword *NORMALIZE* goes, before thirdly the relation or subquery against we do the normalization is specified. Afterwards, we de-

fine the common attributes, which are at least always the timestamps. Finally, the whole operation must be enclosed with parentheses and renamed with some identifier.

- **ALIGN**
  The keyword *ALIGN* is used to execute a temporal alignment on the query. For this, we firstly specify the relation or subquery which shall be aligned against another one. Then, secondly the keyword *ALIGN* goes, before thirdly the relation or subquery against we do the alignment is specified. Afterwards, we define the common attributes, which are at least always the timestamps. Finally, the whole expression must be enclosed with parentheses and renamed with some identifier.

```
SELECT [ALL | DISTINCT [ ON ( expression [ , ...] ) ] ]
    [ [] | CONF | LINEAGE | LINEAGE CONF | CONF LINEAGE ] -- CONFIDENCE
        COMPUTATION
    * | expression [ [ AS ] outputname ] [ , ... ]  --SELECTION (AS DEFAULT)
[ FROM
    [ -- NORMALIZATION
        (from_item1 NORMALIZE from_item2 ON
            from_item1.ts = from_item2.ts AND from_item1.te = from_item2.te [
                AND ...]) renamed_item
    ] | [ -- ALIGNMENT
        (from_item1 ALIGN from_item2 ON
            from_item1.ts = from_item2.ts AND from_item1.te = from_item2.te [
                AND ...]) renamed_item
    ] | [ -- NO TEMPORAL OPERATION (AS DEFAULT)
        from_item
    ]
    [ , ... ]
]
[ WHERE condition ]
...
```

Figure 4.1: PL/pgSQL Synopsis

4.2 Server setup

First of all, a web server had to be established, such that people could use the website after its release. Moreover, the web server had to allow the installation of a specifically modified version of PostgreSQL, a well-known object-relational database management system (ORDBMS). This was required, as temporal probabilistic databases are not yet part of the current PostgreSQL release. Thus, the setup of a web server with web languages and scripts like PHP for server-side processing, HTML, JavaScript and CSS for client-side processing. In addition, the installation of a PostgreSQL-Version with temporal probabilistic database support developed at the Department of Informatics at the University of Zurich was indisputable.

By default, a website with PHP, HTML, JavaScript and CSS can be run as soon as the server is configured as a webserver and the installation of PHP is completed. Yet, Linux servers do not allow for any kind of connection to any PostgreSQL-Server on the same server by default.

According to questions and answers found online, this is due to a bug in the Linux kernel security module called Security-Enhanced Linux or short SELinux. The only solution found to overcome this problem was the disabling of this feature (SELinux). Thereupon, connections to the PostgreSQL-Server could be made without any problems after having set the settings in the *postgresql.conf* file. In this file, specifications about who is allowed, e.g. users of the same network, to establish a connection of which type, e.g. only with login and password, to the PostgreSQL-Server.

4.3 Independent workspaces

As described in section 3.1, our application features independent workspaces. This does not only allow a user to store safety-critical data, but this approach is also more secure against malicious attacks, which could be executed by sending scripts through a query.

In terms of implementation, this is achieved by creating a new login role on the PostgreSQL-server on registration. By default, those login roles are unique and have no permission to create or amend further roles, databases, replications or PostgreSQL-settings. Thus, a user can only create, access and modify the relations, functions and tuples the user creates himself.

Apart from creating a login role for each user, the registration script also creates a unique database for it. This does not only improve the general overview for an administrator to see which data belongs to whom, but users are also less restricted by the names they choose for their functions and relations. Therefore, it cannot occur that another user already blocks a certain relation name, as each user works on his own database.

4.4 Result table

Assuming that the keyword *LINEAGE* is specified in the query, a summarized extract of the algorithm which creates the result table is shown in Figure 4.2. Firstly, the query must be written or selected, before it can be sent to the server. There, we connect to the PostgreSQL database and store the query in the database, before we actually execute the query itself. We then process the process the result before we send it back to the client, where we finally set up the DataTable.

In order to create a DataTable with the DataTables-library, we either need the table content as a JSON-object or an HTML table. As we do not need to make further computations with the result table itself apart from reading out lineage, we decided to parse the query result into an HTML table, which will be transmitted as a string to the client's computer. As a rule of thumb, the same content can be more efficiently transmitted between a server and a client if it is a string rather than a JSON-object due to its size. On the other hand, JSON-objects are very handy, if further computation needs to be made, as they allow for easy access and modifications of their attributes and values. Therefore, given that the DataTables-library supports both approaches, we retrieve the result as an HTML table in order to decrease the amount of data that has to be transmitted and thus time.

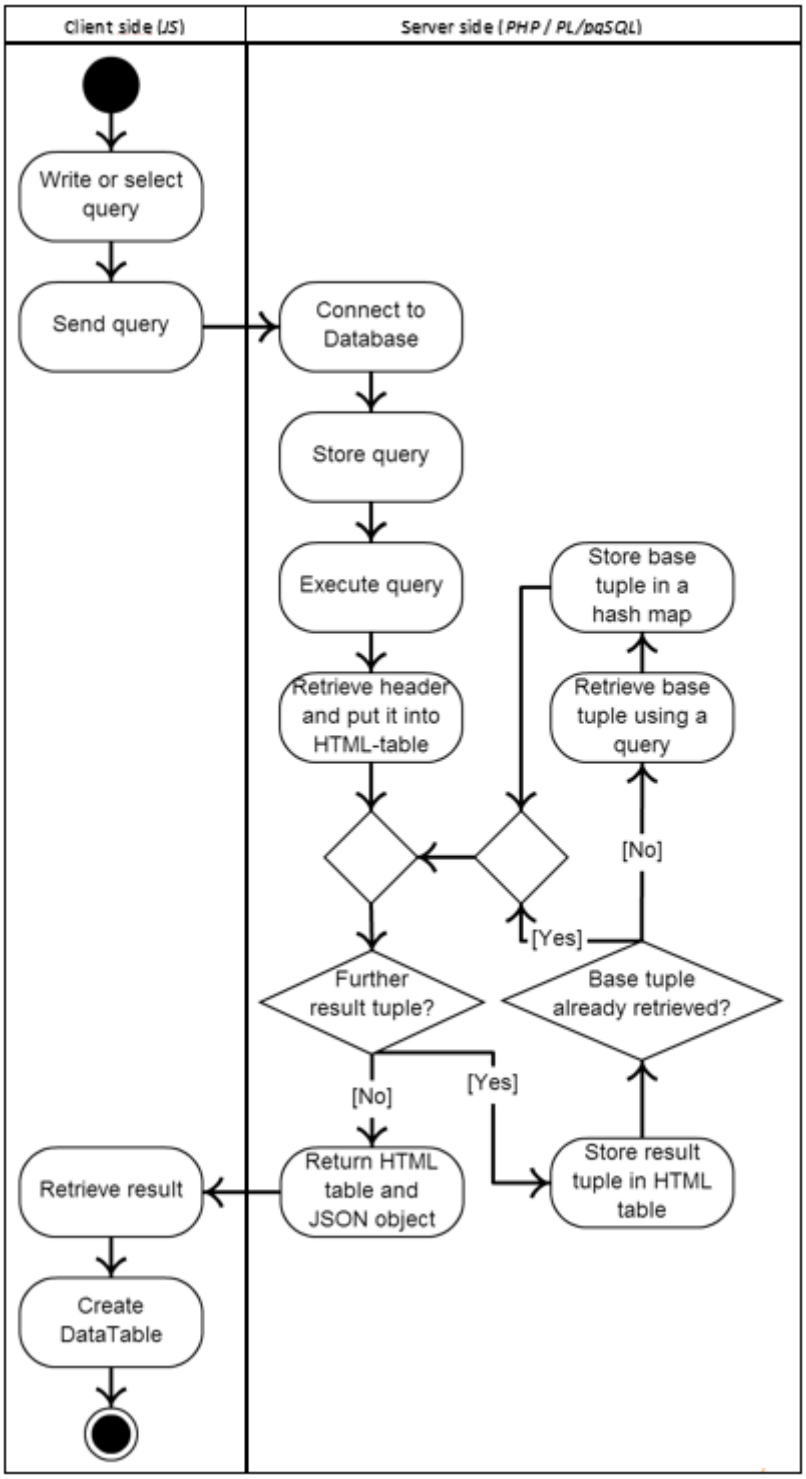

Figure 4.2: Algorithm to create the result table

In the same go, we also retrieve the base tuples as JSON-objects, as we will need them for several occasions:

- In order to compute the range and marginal rates for the bubble chart
- In the bubble chart, when a user clicks on a bubble
- In the lineage tree, when a user clicks on a leaf node

Therefore, instead of connecting to the server, re-querying a base tuple and retrieve the result, we just query all base tuples at once, as we need them all either way, in order to create the bubble chart.

The most efficient approach we have found for this was to parse the lineage expression of every result tuple when parsing it into the HTML table. By analysing the lineage expression, we looked for each base tuple that we found whether it was already stored in our previously created hash map or not. If not, we queried its attributes and values and added it to the hash map. Once having analysed all lineage expressions, we encoded the hash map into a JSON-object and returned it together with the HTML table to the client.

4.5 Visualisation

Assuming that the keyword *LINEAGE* is specified in the query, result tuples can be further investigated by clicking on them. The creation of the bubble chart as well as the lineage tree is summarized in Figure 4.3.

In order to create a lineage tree using the Data-Driven Documents-library, we need to have the Boolean expression (lineage) in form of a JSON-object. For this, we gather for each result tuple the lineage string, which is currently being stored in a hidden column of the displayed DataTable.

The implementation is a recursive function that takes as input a Boolean expression in form of a string and returns a JSON object. However, it remains to note that there are several constraints on the representation of lineage. First of all, all constraints mentioned in the introduction to this section still apply. They are:

- Each child of a Boolean operator must be within parentheses
- No white-spaces
- Siblings may not represent another boolean operator
- $\neg$ is represented by $-$, $\wedge$ by $*$ and $\vee$ by $+$

In addition to those constraints, each base tuple must be followed by the character ';'. In order to guarantee those constraints, we already made some slight changes to the lineage string we receive from the PostgreSQL-server when having parsed it into the resulting HTML table. The benefits of these constraints are that we can parse lineage writing less complex and thus more maintainable source code.

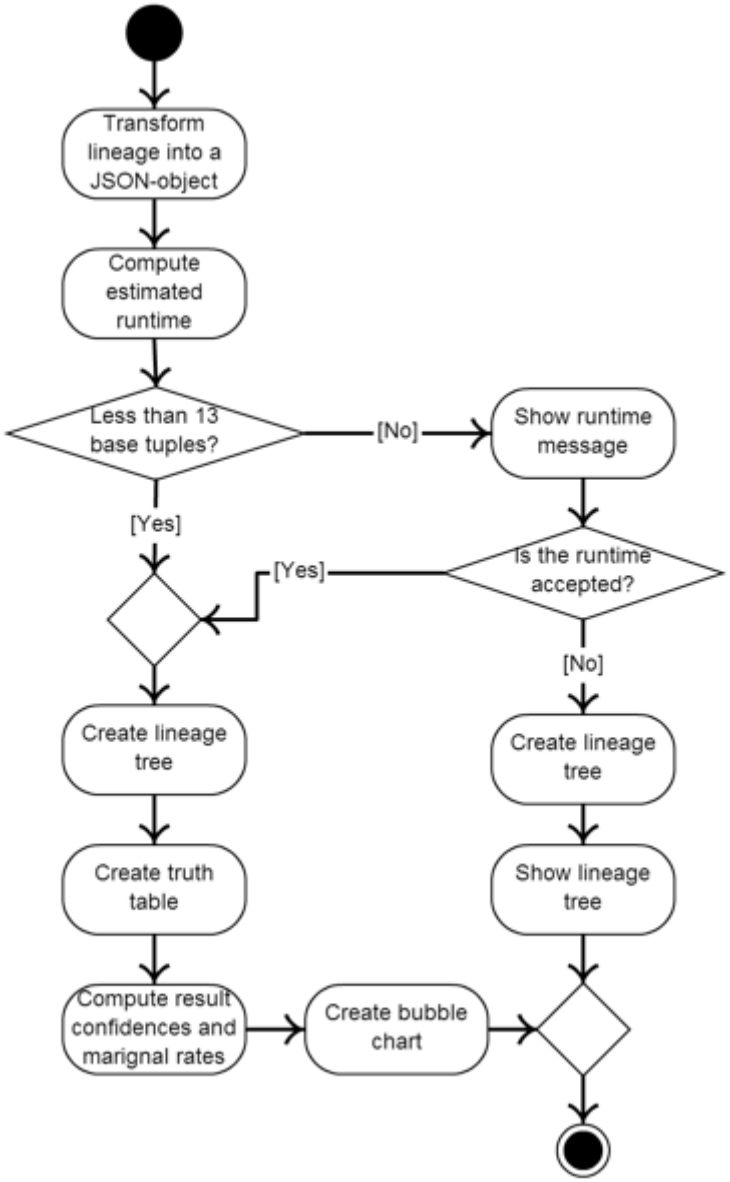

Figure 4.3: Algorithm which creates the bubble chart and the lineage tree

Example 18 *A string according to "(A.1;)+(A.2;)+(A.3;)" would be valid, while "A.1;+A.2;*(A.3;)" is invalid due to lacking parentheses and siblings which represent a different Boolean operator.*

In detail, we analyse each character of the string. If it is an opening parenthesis, we look for its corresponding closing parentheses and recursively call the function on their content. In this case, we will store its result as a child of the current node of the JSON-object. However, if the character represents a Boolean operator, we set the name of the current node of the JSON-object to its symbol. Otherwise, if it is the first character of a base tuple, we create a child of the current node of the JSON-object and set its name to the name of the base tuple reference. The creation of a child for a base tuple is essential, as we otherwise might overwrite the name of an operator.

Example 19 *Consider that the algorithm would not create a new child for a base tuple. Then, parsing lineage "A.1;+A.2;" would result in setting the name of the corresponding node object first to $A.1$, before it will be overwritten with $+$ and eventually set to $A.2$.*

Once having parsed the whole (sub)string, we look whether we created any children for the current node. If so, we investigate whether there are more children than we specified in the threshold, currently being set to 3. If this is the case, too, we add any but three children to one of the two further array objects, which are annotated to the current node in the JSON-object. Their function is to hide any child of a node going beyond the threshold, such that the tree remains straightforward and navigable. In detail, a node will be annotated with at least one triangle, whenever not all children will be displayed in the lineage tree. By clicking on it, the previous or next children will be loaded, whereas the children that disappear will be added to one of the two object arrays we just mentioned before, depending in which direction the user loaded further children.

Apart from that, we also translate the $+$, $*$ and $-$ operands back to $\vee$, $\wedge$ and $\neg$, but represent them in the HTML format. Moreover, we add an attribute pos to a node, such that we can keep track at which child index we are, when traversing the children from one array into another.

Example 20 *Consider lineage "((A.1;)+(A.2;+(A.3;)+(A.4;))*(B.1;)". Then, the corresponding JSON-object is as shown in Figure 4.4.*

The children of the "" operand are "(B.1;)" and "((A.1;)+(A.2;+(A.3;)+(A.4;))". Thus, we set the name of the root node to '∧', before recursively analysing its children. For child "((A.1;)+(A.2;+(A.3;)+(A.4;))", the operands are "+", thus we assign '∨' as the name of this child node. Then, by temporarily storing the four children $A.1$, $A.2$, $A.3$ and $A.4$., we find out that we have more than 3 children. Thus, one child is added to the 'hiddenBelow' array. The other array remains empty and the variable 'pos' will be initialised to 0. Afterwards, we do the same for the other child "(B.1;)" of the root node.*

Given that, we are now able to draw the lineage tree using the Data-Driven Documents-library. Moreover, by analysing how many different base tuples we have in the current lineage, we can make an estimation how long it will take to create the truth table and thus the bubble chart.

```
{
  "name": "&and;",
  "children":
  [
    {
      "name": "&or;",
      "children":
      [
        {"name": "A.1"},
        {"name": "A.2"},
        {"name": "A.3"}
      ],
      "hiddenAbove": [],
      "hiddenBelow":
      [
        {"name": "A.4"}
      ],
      "pos":0
    },
    {"name": "B.1"}
  ],
  "hiddenAbove": [],
  "hiddenBelow": [],
  "pos":0
}
```

Figure 4.4: JSON-object equivalent to "((A.1;)+(A.2;+(A.3;)+(A.4;))*(B.1;)"

If there are less than 13 different base tuples present or if the user wants to compute the bubble chart either way, we create a truth table according to section 2.1. In this regard, as we have lineage now as a JSON-object, we are able to evaluate a set of Boolean values for the JSON-object more efficiently. This as we do not necessarily need to explore each child of the JSON-object. For example, if a node of the JSON-object represents an $\wedge$-operator and if the first child evaluates to *false*, it does not matter to what the other children would evaluate, as the corresponding node will evaluate to false anyway. Similarly for the $\vee$-operator, where we only need at least one child which evaluates to true.

This is a major improvement to the PostgreSQL implementation we use, as we will see in Figure 5.1 in section 5.1. In fact, since the version on which we enhanced PostgreSQL with the support for temporal probabilistic databases did not support JSON-objects or something similar to it, the evaluation must be done by transforming lineage into a postfix notation. This in return means that we cannot leave out some children as in the JSON-object in order to get a correct result. Apart from that, due to the lack of accessing an element at a certain position in an array directly, further iterations need to be made, which all significantly increases the runtime. Therefore, we propose to avoid the keyword *CONF* in queries that generate large lineage expressions, as the result confidence can be computed more efficiently by clicking on a result tuple afterwards. Moreover, by computing the result confidence afterwards we do not need to make it for all result tuples together, but only for the investigated one.

Given the truth table, we proceed with the confidence computation similarly to 2.2. However, we merge it together with the marginal rate and range computation as discussed in section 2.3 in order to avoid further overhead. In detail, we iterate over each base tuple and override its probability with zero and one respectively in order to get the first and second boundary. Then, by further calculations according to section 2.3, we eventually can compute everything we need in order to create the bubble chart using the Data-Driven Documents-library again.

5 Evaluation and Future Work

In section 5.1, we make a detailed runtime evaluation, while we draw concrete conclusions and propose ideas for future work in section 5.2.

5.1 Detailed runtime evaluation

Reconsidering section 4.4 and 4.5, our implementation consist of two main algorithms: One that computes the result table with its lineage and another that creates the visualisation of a result tuple.

Previous analysis have shown that the performance of the result confidence computation can be vastly improved by using another algorithm than creating and evaluating a truth table. However, the given PostgreSQL-implementation did not feature those improved algorithms at this stage. Apart from that, there is further overhead when executing a query as we have discussed in section 4.4.

Earlier evaluations have shown that the runtime of a query on a PostgreSQL server depends on the number of input tuples as well as the operations it contains. However, since our visualisation is made on demand per result tuple, we would rather look on the complexity of a lineage expression than on the number of input tuples. The reason being that many input tuples do not necessarily imply complex lineage expression. For example, a selection will produce many result tuples with simple lineages, whereas a duplicate elimination will produce few result tuples with complex lineages.

In general, the duration of the result confidence computation can be measured in the number of different base tuples the corresponding lineage expression contains. While the length of the lineage expression has an impact on the runtime too, a natural language query is likely to have its bottleneck in the number of different base tuples per lineage expression, due to the exponential algorithm.

Indeed, by looking at Figure 5.1, we can clearly identify that the bubble chart generation as well as the result confidence computation within a query is exponential. Unfortunately, already 10 or more different base tuples in a lineage expression might significantly increase the execution time for both the query execution and the bubble chart generation. Such an event is likely to occur when using duplicate elimination over many duplicates for example or an aggregation that aggregates many tuples together. Under some circumstances, the difference and intersection operators might produce this dilemma as well given that the relations consit of numerous duplicates.

In Figure 5.1, we see that the keyword *LINEAGE* has nearly no impact on the execution time of a query. This as we do not need to create any time-consuming truth table. However, this is only true in the case that each base tuple occurs only exactly once in a lineage expression. Given that a base tuple might be present more than once, the lineage expression gets longer and more characters have to be transmitted from the server back to the client. In general, the more characters we have to transmit, which might be due to long lineage expressions, many

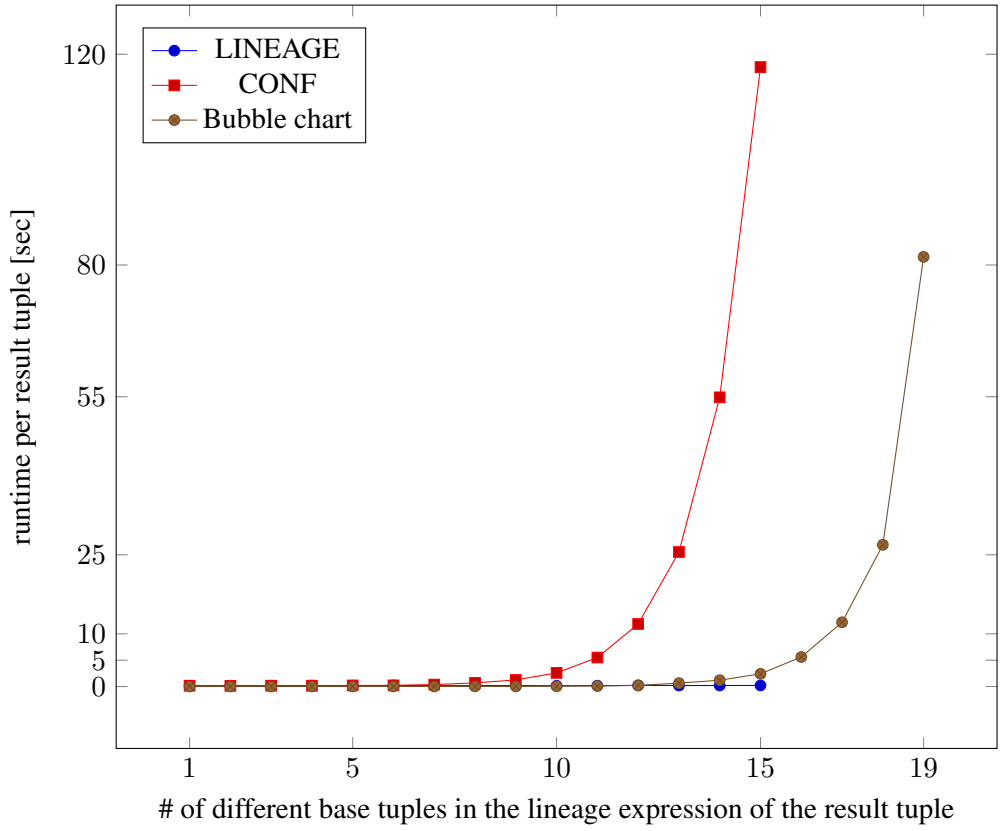

Figure 5.1: Minimum runtime per result tuple

result columns or many result tuples, the longer it takes to receive the result and to generate the DataTable. Thus, apart from choosing a data type which represents as much with as few bytes as possible, we cannot really omit anything. However, research could be done how to rewrite the Boolean expression of rewrite into a shorter format.

It remains to explain why the result confidence computation per result tuple on the PostgreSQL server is worse than when generating a bubble chart and why this algorithm is implemented twice. First of all, reasoning why the algorithm in our web application is quicker although being of the same kind has already been described extensively in section 4.5. Secondly, the algorithm is implemented twice, because the result confidence and the lineage expression is not sufficient to compute the marginal rate of each base tuple on the corresponding result tuple. Therefore, since the PostgreSQL implementation is not yet extended with the calculation of those properties, we need to recreate the truth table and evaluate it. However, it remains questionable whether the implementation of these calculations in PostgreSQL would actually be aspiring, even if the implementation would be updated to a newer version where JSON-objects are supported. This in regard to an approach to compute the impact of a probability change in the focused tuple on all instead of only the corresponding result tuple.

Reconsidering Figure 5.1, it remains to note that creation time of the lineage tree is tremendously fast. Even if the lineage tree consists of 1000 children or more, it is being generated within less than a second. This as we do not have any result confidence computation overhead and so on. That is also the reason why we are able to show the lineage tree at any time, even if the bubble chart generation would take too long.

5.2 Conclusion and Future Work

Our web application allows to represent any natural language query in PL/pgSQL and the execution of it. The result can easily be further investigated and understood due to the interactive result table, the bubble chart and the lineage tree visualisation. Given the predefined datasets and queries, the application also suits to less experienced users and users who want to enhance their understanding of lineage and the impact of the probability of a base tuple on the corresponding result tuple. This however might be improved in the way that we not only compute how the confidence of the corresponding result tuple could be changed, but how it affects the confidences of all other result tuples, which were derived from the focused tuple as well.

A further advantage of our application is that result confidences can be computed on a per result tuple basis, which means that the keyword *CONF* must not be specified unless an overview over the result confidences over all result tuples is wished. Furthermore, our visualisations also handle large lineage expressions without any problem. No matter how many distinct base tuples there are and how long lineage is, we are able to compute the visualisation of the lineage tree efficiently, which itself remains straightforward and navigable due to the collapsible and expandable nodes. Moreover, using a threshold of three children per node, we guarantee that a node expansion will not result in a chaos nobody can understand.

Assuming that a lineage contains many different base tuples a warning will be shown about how long it might approximately take to compute the bubble chart on a normal computer. Thereupon the user can either accept or deny the computation. Thus, we avoid that the browser of the user unintentionally gets stuck during the computation of the bubble chart.

Apart from implementing more efficient algorithms to compute exact or approximate result confidence computation algorithms, an interesting feature would be a mask, where user can write their query by clicking on relations and relational algebra operators, without having to actually think how to map that into PL/pgSQL.

References

[1] Anton Dignös, Michael H Böhlen, and Johann Gamper. Temporal alignment. In ACM SIGMOD 2012 international conference on Management of Data, SIGMOD '12, pages 433–444, New York, NY, USA, MAY 2012. ACM.

[2] Robert Fink, Dan Olteanu, and Swaroop Rath. Providing support for full relational algebra in probabilistic databases. In Proceedings of the 2011 IEEE 27th International Conference on Data Engineering, ICDE '11, pages 315–326, Washington, DC, USA, 2011. IEEE Computer Society.

[3] Anish Das Sarma, Martin Theobald, and Jennifer Widom. Exploiting lineage for confidence computation in uncertain and probabilistic databases. Technical Report 2007-15, Stanford InfoLab, March 2007.