Probabilistic Databases. MayBMS

Anas Al Bassit
Katsiaryna Krasnashchok

Brussels, 2015

Table of contents

Introduction

Managing large data sets has become a key feature of many applications in the last two decades. In addition, more and more often applications dealing with real-life data need to handle uncertainty in it. There are vast variety of reasons for imprecision in data: in sensor and RFID data imprecision happens due to errors in measurement; in information extraction it comes from the ambiguity of natural-language text; and in business intelligence imprecision is used to reduce the cost of data cleaning (1). In some areas, such as privacy-preserving data mining, additional errors may be added intentionally in order to mask the identity of the records so this data can later be published and used for research (2). Decision support and diagnosis systems naturally use hypothetical queries. Scientific databases for storing the results of scientific experiments, frequently contain uncertain data such as incomplete observations or imprecise measurements. Software in the fields of fighting crime or terrorism, tracking moving objects, and plagiarism detection essentially relies on techniques for processing and managing large uncertain datasets (3).

Traditional database management systems are not designed to deal with uncertain data, which is why these new progressive applications quickly became so popular: they query, search, and aggregate large volumes of imprecise data to find the "diamonds in the dirt" (1). This tendency points to the growing need for universal tools to handle imprecise data. Here, we offer an overview of the state of the art techniques to handle imprecise data, in particular, probabilistic data.

A probabilistic database management system – ProbDMS – is a system that stores large volumes of probabilistic data and supports complex queries (1). In addition, a ProbDMS needs to also support regular types of queries, such as updates or inserts, just like any database management system (DBMS) does. Major challenge in a ProbDMS is that it needs to provide both handling of probabilistic information and scalability. And while many scalable data management systems exist, probabilistic inference is still a complex issue, and current systems do not scale to the same extent as data management systems do (1).

Artificial Intelligence research community has made much progress in the field of inference in uncertain data in the past years. Some of the most promising AI projects belong to this area, for example implementation and application of graphical models in biology. While a number of papers on uncertain data and probabilistic databases have been already written over the past decades, this area has become the focus of research interest very recently, and work on scalable systems has only just started (3). This tendency proves that there are many further potential applications of probabilistic data management, which will emerge in future when probabilistic database systems become fully available for use in real-world situations.

In this report, we describe the key concepts in probabilistic database systems. Furthermore, we offer an overview of one of the most popular existing products – MayBMS, and demonstrate its possibilities on a case study.

Overview of probabilistic databases and management systems

An example of probabilistic database

The best way to start a journey of learning new theory or technology is a demonstration. Let us illustrate the use of probabilistic information on an example from an information extraction system.

Example 1. We constructed a simple skeleton of a database for a system that consists of structured information about the football players. All data is found on the Web: the system extracts lists of players together with related information about them. Figure 1(a) illustrates the players' (possible) current team assignments, and Figure 1(b) illustrates the roles they (might) have taken in different championships. Each tuple contains the probability of it being a true fact.

9 tuples in Players are grouped in four groups of disjoint events, e.g. t_2^1, t_2^2, t_2^3 are disjoint, and so are t_3^1, t_3^2. And tuples from different blocks are independent, e.g., t_1^1, t_2^2, t_4^1; also 11 tuples in Championships are independent probabilistic events. X and Y are hidden variables used for the examples in the following sections.

	Name	**Team**	**P**	
t_1^1	Messi	Barcelona FC	$P_1^1 = 1$	$X_1 = 1$
t_2^1	Ronaldo	Manchester United	$P_2^1 = 0.3$	$X_2 = 1$
t_2^2		Real Madrid	$P_2^2 = 0.6$	$X_2 = 2$
t_2^3		Sporting Lisbon	$P_2^3 = 0.1$	$X_2 = 3$
t_3^1	Neymar	Santos	$P_3^1 = 0.2$	$X_3 = 1$
t_3^2		Barcelona FC	$P_3^2 = 0.8$	$X_3 = 2$
t_4^1	Figo	Real Madrid	$P_4^1 = 0.5$	$X_4 = 1$
t_4^2		Barcelona FC	$P_4^2 = 0.4$	$X_4 = 2$
t_4^2		Sporting Lisbon	$P_4^3 = 0.1$	$X_4 = 3$

Figure 1 (a) Players

	Name	Championship	Role	P	
s_1	Messi	Champions League	Striker	$q_1^1 = 1$	$Y_1 = 1$
s_2	Messi	La Liga	Right Wing	$q_2^1 = 0.3$	$Y_2 = 1$
s_3	Messi	La Liga	Striker	$q_2^2 = 0.6$	$Y_3 = 1$
s_4	Messi	La Liga	Play Maker	$q_2^3 = 0.1$	$Y_4 = 1$
s_5	Ronaldo	Champions League	Striker	$q_3^1 = 0.2$	$Y_5 = 1$
s_6	Ronaldo	La Liga	Left Wing	$q_3^2 = 0.8$	$Y_6 = 1$
s_7	Ronaldo	La Liga	Striker	$q_3^2 = 0.2$	$Y_7 = 1$
s_8	Neymar	Brasileiro Série A	Left Wing	$q_4^1 = 0.2$	$Y_8 = 1$
s_9	Neymar	La Liga	Left Wing	$q_4^2 = 0.7$	$Y_9 = 1$
s_{10}	L. Figo	Champions League	Left Wing	$q_{10} = 0.6$	$Y_{10} = 1$
s_{11}	L. Figo	La Liga	Right Wing	$q_{11} = 0.2$	$Y_{11} = 1$

Figure 1 (b) Championships

As we can see, the extracted teams of the players in Figure 1(a) are not unique. These variations occur because of outdated or false information that we can often find on the Web. But even if we are operating on the most recent webpage, the extraction is a difficult process, so we have to produce many alternative results in order not to miss valuable data (1). As a result, each Name contains several possible teams. Therefore, we can think of Team as an attribute with imprecise values. There are two constraints on the data of this example: tuples with the same Name but different Team are mutually exclusive; and tuples with different values of player's Name are independent. The championships and roles in Figure 1(b) are extracted from sport-themed websites and are also imprecise. In both examples, the uncertainty is showed as a probabilistic confidence score, which can be computed, for example, by the extracting program. There can be other kinds of uncertainties, for example, probabilities produced after applying entity matching algorithms (does Neymar in one webpage refer to the same person as Neymar in another webpage? We cannot assume that there is only one player with such name in football world). The example in Figure 1 demonstrates a general principle: uncertain data has a confidence score (i.e. probability) attached to it (1).

Probabilistic graphical models

Before we can start talking about probabilistic database management systems, it makes sense to define the models that are commonly used to represent probabilistic data.

Probabilistic graphical models is a powerful class of approaches that enable us to compactly represent very large joint probability distributions. They deal with the uncertainty by combining probability theory graph theory and have been proven useful in a wide range of domains, such as natural language processing, social networks, bioinformatics, code design, sensor networks etc. (4).

A graphical model consists of two components (2): 1) A graph whose nodes are the random variables and whose edges connect variables that interact directly. 2) A set of small functions called factors - each over a subset of the random variables. The set of factors that can be associated with a graphical model can be undirected or directed and differ depending on the structure of the graph. The power of graphical models comes from the graphical representation of factors that makes them easy to understand. There are two popular classes of graphical models: Bayesian networks (directed models), and Markov networks (undirected models) - depending on the nature of the interactions between the variables (2).

Bayesian networks

Bayesian networks - directed graphical models, are used to represent interactions between random variables. Directed edge from variable X_i to variable X_j in the acyclic graph means that X_i directly influences X_j. Moreover, if X_i is not a descendant or a parent of X_j , then $X_i \perp X_j \mid parents(X_j)$. The probability distribution that a directed graphical model represents is:

$$Pr(X_1, \ldots, X_n) = \prod_{i=1}^{n} Pr(X_i \mid parents(X_i)).$$

This means that each of the factors associated with a Bayesian network is a conditional probability distribution over a node given its parents in the graph (5).

Example 2. While having an evening walk in the park, Mr. Holmes gets a phone call from Dr. Watson, his neighbor, who claims that there is a burglar alarm sound coming from the direction of Mr. Holmes's house. Mr. Holmes, obviously, starts to worry and prepares to rush home, but at the same time he recalls that Dr. Watson is known in the neighborhood as the practical joker. So he (Mr. Holmes) decides to call his other neighbor first – Mrs. Gibbon, who he considers more reliable despite her drinking problem (4). The graph representing this uncertainty model (where "Alarm sound" is the uncertain variable) looks like this:

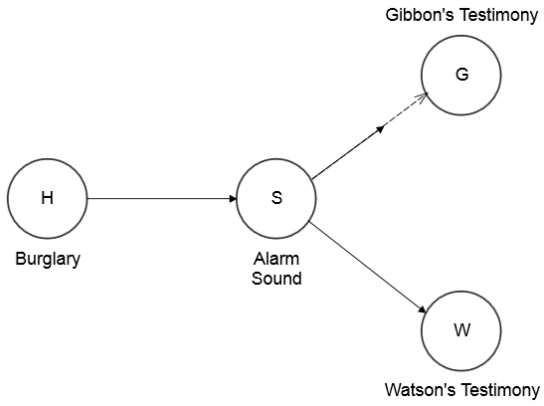

Figure 2 Bayesian network graph

Since Bayesian networks are easy to design, interpret and reason about, they are extensively used in practice (2)**.**

Markov networks

Undirected graphical models - Markov Networks, are useful for representing distributions over variables where there is no natural direction of the influence of one variable over another. Examples include atoms in a molecular structure, the labels of pixels of an image, or the interactions between environmental sensors data (2).

The probability distribution represented by a Markov network factorizes in a somewhat less intuitive manner than Bayesian networks. Let G be the undirected graph over the random variables $X = \{X_1, \ldots, X_n\}$ corresponding to a Markov network, and let CS denote the set complete subgraphs of G. Then the probability distribution represented by the Markov network factorizes as follows (6):

$$Pr(X_1, \ldots, X_n) = \frac{1}{Z} * \prod_{C \in CS} f_C(X_C)$$

where $f_C(X_C)$ are the factors over a complete subgraph of G. $Z = \sum_X \prod_{C \in CS} f_C(X_C)$ is the normalization constant.

Example 3. A very simple example with random variables describing the tuberculosis status of four patients. Patients that have been in contact are linked by undirected edges. Each edge means the possibility for disease transmission. For example, Patient 1 has been in contact with both Patient 2 and Patient 3, but not with Patient 4 (6).

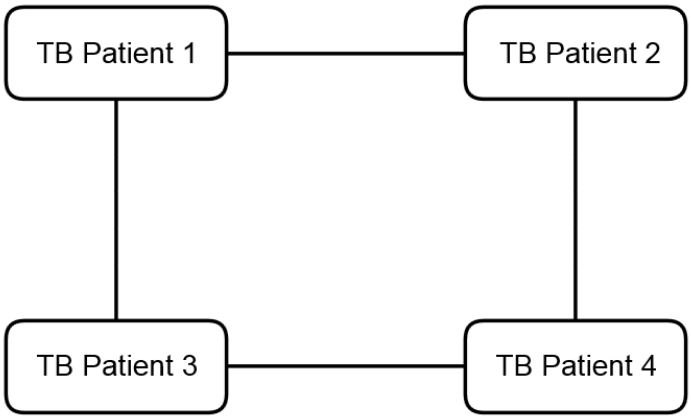

Figure 3 Markov network graph

The conditional independences captured by a Markov network are determined as follows: if a set of nodes X separates sets of nodes Y and Z, then Y and Z are conditionally independent given X.

Characteristics of a ProbDMS

There are three most important characteristics of any ProbDMS (1):

1) How do we store (represent) a probabilistic database?
2) How do we answer queries using the chosen representation?
3) How do we present the result of queries to a user?

Representation. There is a conflict between the complexity of representation and the possibility to scale the system. A simple way of representation where each tuple is an independent event with probability attached is easier to process, but it cannot model the correlations between events. In contrast, if we take a more complicated representation, e.g., a large Markov Network (6), it can capture the semantics of the data, but it may be impossible to compute even simple SQL queries using this model. Additional problem is ensuring that the representation system should is applied to relational database – for handling non-probabilistic data (1).

A ProbDMS needs to support complex decision-support SQL. Simple queries may be useful for some relatively small applications or test examples, but the real value comes from queries that work with large number of tuples, or aggregate over big amount of data. For example, the result of the query to find the Team of the Play Maker in "La Liga" will be obviously imprecise and relatively meaningless, but a query such as finding all teams with more than 6 "La Liga" members returns much more interesting and useful results.

Query computing. There are two logical steps in computing any SQL query on probabilistic data sets (1): first, fetch and transform the data, and second, perform probabilistic deduction. The first approach that comes to mind is to separate them: use a database engine for the first step, and some probabilistic inference algorithm for the second. But on large data sets the probabilistic deduction process quickly becomes much more time consuming than the actual query. So a better approach is to integrate the steps: this makes it possible to use some database specific techniques, such as query optimization, materialized views and metadata.

User interface. The result of an SQL query is a set of tuples. The challenge of any ProbDMS is to represent this set to the user in such a way that the correlations between the tuples are clear. Additionally, a big problem is the obtaining feedback from the users and using this feedback to "clean" the database. This is a difficult and complex issue, which has not yet been solved (1).

Applications of probabilistic database systems

Probabilistic databases are currently being used (or potentially can be used) in a wide range of applications. Information obtained from sensors that acquire temperature, pressure, or humidity data from the environment - the BBQ system (7) showed that this sensor data can be managed with a probabilistic data model. In this case probabilistic model can answer wide range of queries without needing additional data from the sensor. This proved to be an important optimization since fewer sensor readings basically means longer battery life (7).

In data cleaning, deduplication is one of the key components. It was proven that a probabilistic database can simplify the task of finding and removing duplicates, by allowing multiple conflicting tuples in the database with probabilities attached to them (8).

Many other applications have looked may use or already use probabilistic databases for their data management needs: social networks, management of anonymized data, scientific data management (1), crime fighting, surveillance, tracking moving objects, plagiarism detection, predicting terrorist action, lean expert systems and decision support systems (9).

Semantics of a probabilistic database: possible worlds

The formal semantics of a probabilistic database is the "possible worlds model" (10). In general, a probabilistic database is a probability space over the possible contents of the database (1). If there is a single table in the database instance (I), it is simply a set of tuples from the table, i.e. this is a traditional database. A probabilistic database is a discrete probability space:

$$PDB = (W, P)$$

Where $W = \{I_1, I_2, \ldots, I_n\}$ is a set of possible instances - possible worlds;

And $P : W \rightarrow [0, 1]$ is such that

$$\sum_{j=1}^{n} P(I_i) = 1.$$

So there is one random variable for each possible tuple having values of 0 (the tuple is not present) or 1 (otherwise), and a probabilistic database is a joint probability distribution over the values of these random variables. This is a very general and powerful definition, which covers all the data models that have been studied so far (1).

Consider some tuple t: the probability that the tuple belongs to a randomly chosen world is

$$P(t) = \sum_{j:t \in I_j} P(I_j)$$

which is called the **marginal probability** of the tuple t. Furthermore, if we have two tuples t_1, t_2, we can figure out the probability that both are present in a randomly chosen world - $P(t_1 t_2)$. When the this probability equals to $P(t_1)P(t_2)$, we can say that t_1 and t_2 are independent. If it equals to zero, then we say that t_1 and t_2 are disjoint (or exclusive) tuples. If none of these is true, then the tuples are correlated in some other, not obvious way (1).

If we now consider a query $\boldsymbol{Q}$, and a possible tuple $\boldsymbol{t}$ in the query Q's answer, $P(t \in Q)$ is the probability that, in a randomly chosen world, t is a result of the query Q. This means that a probabilistic database system answering any given query Q should return all possible tuples t_1, t_2 . . . together with their probabilities $P(t_1 \in Q)$, $P(t_2 \in Q)$, etc.

Representation solutions for probabilistic databases

In practice, it is impossible enumerate all possible worlds, so we need to use some more applicable representation techniques. The most popular approach is to allow the possible records to be either

independent or disjoint. A probabilistic database **block** is called **independent-disjoint**, or BID, if we can divide all possible tuples into blocks in such a way that tuples from the same block represent disjoint events, and tuples from distinct blocks are independent (1). This approach was illustrated in Figure 1. The blocks are obtained by grouping Players by Name, and Championships by (Name, Championship, Role). The probabilities are given in attribute P. The tuples t_2^2 and t_2^3 are disjoint (they are in the same block), while the tuples t_1^1 , t_2^2, s_1, s_2 are independent (different blocks).

Sometimes complex applications require a representation technique that can express complex correlations between records. Several techniques have been described in the literature: lineage-based (11), World-Set-Decompositions (12) and **U-relations** (13) (which we will explain and use later in the report). Amongst others are the Probabilistic Relational Model (14), which separates the data from the probabilistic network, Markov Networks, and Markov Chains (6). However, such representations are often hard to understand, and they increase the complexity of query evaluation (1).

Lineage and c-tables notion

The **lineage** of a tuple is an annotation that defines its derivation (1). It is used both to represent probabilistic data, and to represent query results. Lineage is also a powerful technology for understanding and resolving uncertainty in data. Users can provide very detailed feedback if they can see data lineage. And the result can be used to identify unreliable sources of data (1).

The notion of lineage is used together with c-tables - a technique for representing incomplete databases. Here we illustrate c-tables and lineage by using the example in Figure 4. In a c-table, for each tuple there is a lineage of the tuple – a Boolean expression over hidden variables.

Example 4. In our example (continuation of Example 1) there are four tuples: *Barcelona FC*, *Real Madrid*, *Manchester United* and *Sporting Lisbon*, each annotated with a lineage expression over variables $X_1, X_2, X_4, Y_1, Y_2, Y_5, Y_{10}$. The variables in the expression define the world consisting of exactly those tuples whose lineage is true for those variables. So the c-table represents the set of possible worlds defined by all possible assignments of the variables.

Teams	
Barcelona FC	$X_1 = 1 \wedge Y_1 = 1 \vee X_1 = 1 \wedge Y_2 = 1 \vee X_4 = 2 \wedge Y_{10} = 1$
Real Madrid	$X_2 = 2 \wedge Y_5 = 1 \vee X_4 = 1 \wedge Y_{10} = 1$
Manchester United	$X_2 = 1 \wedge Y_5 = 1$
Sporting Lisbon	$X_2 = 3 \wedge Y_5 = 1 \vee X_4 = 3 \wedge Y_{10} = 1$

Figure 4 C-table example

Lineage is a powerful technique used in ProbDMS because of a very important and essential **closure** property (1): the answer to a query over a c-table can always be represented as another c-table, using

the same hidden variables. We illustrate this property on the database in Figure 1, where each tuple has a very simple lineage (just one variable). Consider now the SQL query in Figure 5(a), which finds the team affiliations of all people who played in "Champions League". The answer to this query equals exactly the c-table shown in Figure 4.

Query evaluation

Query evaluation is one of the hardest technical challenge in a probabilistic data management system. Several approaches exist for dealing with query evaluation and probabilistic inference on the lineage expression. One way is to separate them, another approach is to integrate the probabilistic deduction process with the query computation step. The advantage of the latter is that it allows to successfully use standard data management techniques to optimize the probabilistic inference (1).

Certain queries can be evaluated on a probabilistic database by pushing the probabilistic inference part completely inside the query plan. Hence, for these queries the output probabilities are computed during query processing. Queries allowing this are called **safe queries**, and the plan that calculates the resulting probabilities correctly is a **safe plan** (1).

In relational queries users specify what they want. The system translates the query into relational algebra. The resulting expression is called a relational plan and represents how the query is evaluated. Any safe plan allows probabilities to be computed in the relational algebra, by extending its operators to manipulate probabilities (15). The simplest way is to assume that all tuples are independent: a join of two tuples computes the new probability as $p_1 p_2$, and a duplicate elimination that replaces *n* tuples with one tuple computes the output probability using formula

$$1 - (1 - p_1) \cdots (1 - p_n)$$

A safe plan is a plan in which all these operations are correct. The query optimizer ensures the correctness, by using a static analysis on the plan.

Example 5. Let us illustrate with the queries and plans in Figure 5.

```
SELECT x.Team, confidence()
FROM Players x, Championships y
WHERE x.Name = y.Name
and y.championship = 'Champions League'
GROUP BY x.Team
```

Figure 5 (a)

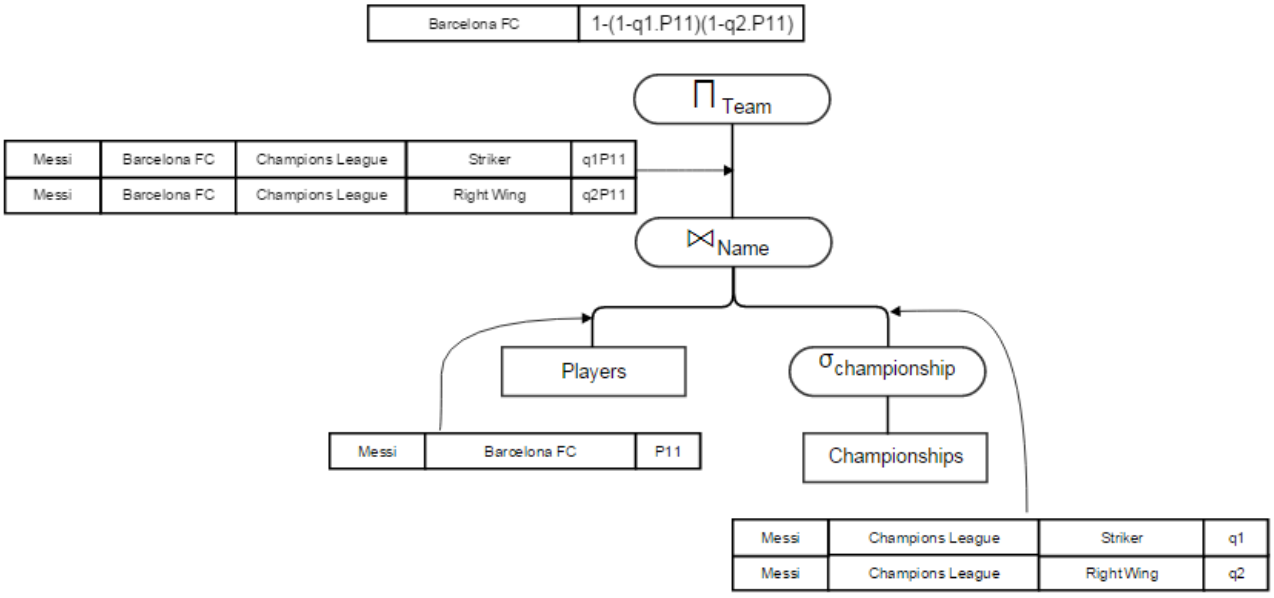

Figure 5 (b)

```
SELECT x.Team, 1-prod(1-x.P*y.P)
FROM Players x,
(SELECT Name, 1-(1-prod(P))
FROM Championships
WHERE championship = 'Champions League'
GROUP BY Name) y
WHERE x.Name = y.Name
GROUP BY x.Team
```

Figure 5 (c)

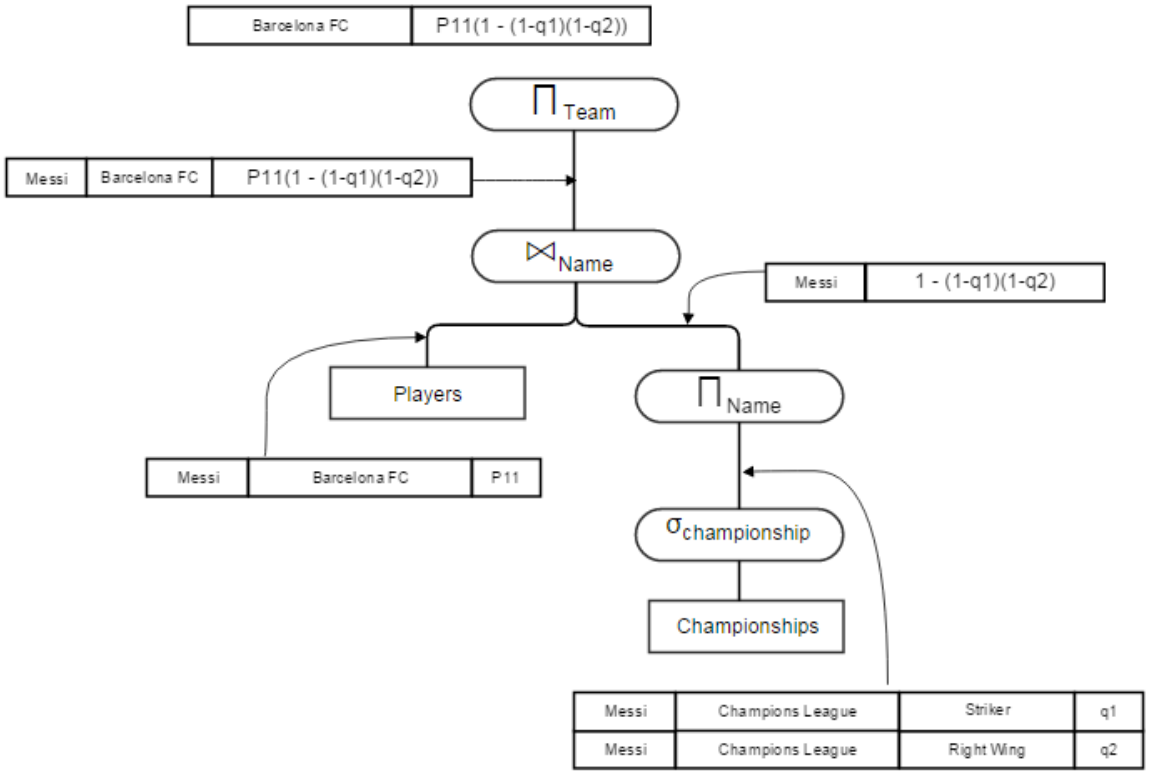

Figure 5 (d)

Figure 5: A SQL query on the data in Figure 1 returning the teams of all players who participated in "Champions League". The query is written in MayBMS syntax: conf() is an aggregate operator returning the resulting probability. The figure shows an unsafe plan in (b) and a safe plan in (d), and also shows the computation of the output probability of Barcelona FC: assuming there is a single player Messi in this team. The safe plan in pure SQL is shown in (c).

Any modern relational database engine will translate the query into the logical plan (b). But this plan is not safe, because the operation $\prod Team$ combines correlating tuples, so the resulting probabilities will be incorrect. The figure illustrates this for the output value *Barcelona FC*: the output probability is

$$1 - (1 - p_1^1 q_1)(1 - p_1^1 q_2)$$

However, the lineage of *Barcelona FC* is

$$X_1 = 1 \wedge Y_1 = 1 \ \vee \ X_1 = 1 \wedge Y_2 = 1 \ \vee \ X_4 = 2 \wedge Y_{10} = 1,$$

so the correct probability is

$$p_1^1 \ (1 \ - \ (1 \ - \ q_1)(1 \ - \ q_2)).$$

Now let us consider the plan shown in (d), which does an early projection and duplicate elimination on Championships. It is logically equal to the plan in (b) logically: this is demonstrated for the same output value, *Barcelona FC*. However, even if plans (b) and (d) are logically equivalent, they are not equivalent in the probabilities computation. Safety is an essential property in query processing on probabilistic databases: because of this, query optimizer needs to search not just for a plan with lowest cost, but for one that is safe. A safe plan can be executed directly by a database engine with slight changes to the implementation of the relational operators (2). Alternatively, a safe plan can also be executed on a regular database engine (of course, for that it needs to be rewritten in pure SQL). Figure 5 (c) shows the safe plan converted into SQL.

When most of the time a safe plan combines calculation of the probabilities to the query plan, it has been shown that is it possible to separate them at runtime (16): the optimizer can choose any query plan (safe or not), and after, the probabilistic inference is handled using the information from the safe plan. This can result in significant execution speedup for many types of queries (1).

Materialized views

Using materialized views for answering queries is quite popular and powerful technique. Here is how it works: there is a number of materialized views, such as answers to previous queries, so the query is rewritten to use of these views, which improves performance. With probabilistic databases, materialized views can have a dramatic impact on query evaluation (1).

However, there is an important issue in using materialized views to handle probabilistic data: the view's output needs to be represented in some way. It is always possible to compute the lineage of all the tuples in the view, but this way we complicate the probabilistic inference. Instead, we want to use only the marginal tuple probabilities. For example, if all tuples are independent probabilistic events, so we only need the marginal probabilities; we can say in this case the view can be fully represented. Of course, generally, not all tuples are independent, but we always can partition the tuples into independent blocks. Furthermore, there will always be the best partition. This is called **partial representation**. Note that the correlation between safe/unsafe views and representable/unrepresentable views is not defined: examples exist for all combinations (17).

Aggregating uncertain data

There are two forms of aggregates in SQL: value aggregates (e.g. for each company return the sum of the profits in all its units), and predicate aggregates (e.g. find the companies having the sum of profit

greater than 1 million euros). Both types are very useful in probabilistic databases. The first type represents an expected value, and the second type of aggregates uses HAVING clause (2).

Most aggregates can be easily calculated. The complexities of computing sum and count, for example, are the same as the complexities of finding the result of the exact same query without the aggregate function (18). Complexity of min and max are same as of the queries with the aggregates replaced by projections with removing columns (18). Average, however, is one of the more difficult aggregates, but it is an important aggregate function for analysis of imprecise data, since it computes the average expectation value. It is hard to calculate even on one table: its expected value is not sum / count(*) of expected values. Some techniques and algorithms have been introduced for aggregates computation (including predicates) (1) which we will not discuss in this report.

MayBMS - an uncertain database management system

Before we start the introduction of MayBMS system, let us briefly list its "colleagues". Research on probabilistic databases is about 30 years old (judging by the first books in this domain). Today there is a lot of interest, mainly driven by the demand from a variety of applications. However, despite this long history and recent interest, currently there are no complete and usable probabilistic database system, the core reason being the computational challenge of combining relational queries with probabilities.

There are several products specializing in probabilistic database management. Most of them are ongoing university researching projects, such as (2):

- The MayBMS project at Cornell University: will be discussed in details in this section;
- The MystiQ project at the University of Washington: lacks sufficient documentation. The goal of the project is to develop efficient query processing techniques for finding answers in large probabilistic databases;
- The Orion project at Purdue University: provides built-in support for uncertainty at the database level (extension of PostgreSQL engine);
- The Trio project at Stanford University: based on an extended relational model called ULDBs, and supports a SQL-based query language called TriQL;
- The BayesStore project at the University of California, Berkeley: again, lacks sufficient documentation; Represents model and evidence data as relational tables; implements inference algorithms efficiently in SQL; adds probabilistic relational operators to the query engine; optimizes queries with both relational and inference operators;
- The PrDB project at the University of Maryland, College Park: based on the notion of "shared factors", supports a declarative SQL-like language for specifying uncertain data and the correlations among them; Also supports exact and approximate evaluation of a wide range of queries including inference queries, SQL queries, and decision-support queries.

In this report we chose MayBMS system to illustrate the capabilities of probabilistic databases. This is an open-source project, one of the most popular products today, with clear goals and comprehensive documentation. MayBMS SQL syntax is based on PostgreSQL (with some additions), which makes it basically ready-to-use for anyone who is familiar with SQL.

MayBMS project

MayBMS is a state-of-the-art probabilistic database management system that has been built as an extension of Postgres, an open-source relational database management system (9) (the source code is available under the BSD license at http://maybms.sourceforge.net).

The system has been under development since 2005 at Cornell University. While the development has been carried out in an academic environment, the creators have always aimed to build a robust, scalable system that can be reliably used in real applications.

Main goals of the MayBMS project (19):

- Create a scalable probabilistic DBMS: representation and storage mechanisms;
- Query and data manipulation language (like SQL) for probabilistic databases;
- Efficient query processing techniques;
- Updates and concurrency control;
- Database APIs for uncertain data;
- Deploy in modern and new data management applications;

MayBMS is a complete probabilistic database management system that supports a powerful, compositional query language. In summary, MayBMS currently has the following features:

- Full support of all features of PostgreSQL 8.3.3, including unrestricted query functionality, query optimization, APIs, updates, concurrency control and recovery, etc.
- Essentially no performance loss on PostgreSQL 8.3.3 functionality: After parsing a query or DML statement, a fast syntactic check is made to decide whether the statement uses the extended functionality: if not, the executed code is exactly the code of PostgreSQL 8.3.3.
- Support for efficiently creating and updating probabilistic databases.
- A powerful query and update language for processing uncertain data that extends SQL with a number of language constructs.
- State-of-the-art efficient techniques for exact and approximate probabilistic inference.

Relational algebra

The query algebra of MayBMS - probabilistic worldset algebra (probabilistic WSA) - consists of a number of commands in addition to regular database (SQL) operations. Here we list the operations with definitions and supporting examples.

1. **The operations of relational algebra** are selection σ, projection π, product $\times$, union $\cup$, difference $-$, and attribute renaming ρ. These operations are executed for each possible world separately and independently. Selection conditions are Boolean combinations of atomic conditions. Arithmetic expressions may occur in atomic conditions and in the arguments of π and ρ. For example,

$$\rho A + B \rightarrow C(R)$$

in each world calculates the sum of A and B values of each tuple of R and saves them in a new attribute C (2).

2. **Conf -** the operation for calculating tuple confidence, i.e. *it computes, for each possible tuple, the sum of the weights of the possible worlds in which it occurs* (19).

   Syntax:

   ```
   conf(R)
   ```

   Semantics on probabilistic database W:

   $$[[conf(R_l)]](W) := \{\langle R_1, \ldots, R_k, S, p\rangle \mid \langle R_1, \ldots, R_k, p\rangle \in W\}$$

   where, $P \notin sch(R_l)$, and

   $$S = \{\langle \vec{t}, P : Pr[\vec{t} \in R_l]\rangle \mid \vec{t} \in \bigcup_i R_l^i\},$$

   with schema $sch(S) = sch(R_l) \cup \{P\}$.

   The result of $conf(R_l)$ - relation S - is the same in all possible worlds, which means it is a certain relation. By the definition of probabilistic databases used in MayBMS, each possible world has probability more than 0. Hence, conf does not return tuples with zero probability.

   Example 6. On a simple example of probabilistic database consisting of 3 possible worlds:

R^1	A	B	
	a	b	$p = 0.3$
	b	c	

R^2	A	B	
	a	b	$p = 0.2$
	c	d	

R^3	A	B	
	a	c	$p = 0.5$
	c	d	

$conf(R)$ results in

$conf(R)$	A	B	P
	a	b	0.5
	a	c	0.5
	b	c	0.3
	c	d	0.7

Moreover, there are two more operations that can be expressed using conf - the possible/certain tuples:

$$possible(R) := \pi_{sch(R)}(\sigma_{Conf>0}(conf(R)))$$

$$certain(R) := \pi_{sch(R)}(\sigma_{Conf=1}(conf(R)))$$

3. **Repair-key -** an uncertainty-introducing operation, which can be thought of as sampling a maximum repair of a key for a relation. Repairing a key of a relation R means to compute (as possible worlds) *all subset-maximal relations obtainable from R by removing tuples such that a key constraint is satisfied* (9). MayBMS uses this operation as a method for constructing probabilistic databases, with probabilities computed from the weights attached to the tuples of R.

   By definition, relation R′ is a *maximal repair* of a functional dependency (fd) for relation R if R′ is a maximal subset of R which satisfies that functional dependency, i.e., a subset $R' \subseteq R$ that satisfies the fd such that there is no relation R'' with $R' \subset R'' \subseteq R$ that satisfies the fd (2).

   Let $\overrightarrow{A}, B \in sch(R_l)$. For each possible world $\langle R_1, \ldots, R_k, \ p \rangle \in$ W, let column B of R contain only numerical values greater than 0. Finally, let R_l satisfy the functional dependency $(sch(R_l) - B) \rightarrow sch(R_l)$. Then,

$$\left[\left[repair - key_{\overrightarrow{A}\ @\ B}(R_l)\right]\right](W) := \{\langle R_1, \ldots, R_k, \pi_{sch(R_l)-B}(\hat{R}_l), \ \hat{p} \rangle \mid \langle R_1, \ldots, R_k, p \rangle \in W, \ \hat{R}\,l \text{ is a maximal repair of fd } \overrightarrow{A} \rightarrow \text{sch}(R_l), \ \hat{p} = p * \prod_{\vec{t} \in \hat{R}_l} \frac{\vec{t}.B}{\sum_{\vec{s} \in R_l:\ \vec{s}.\overrightarrow{A} = \vec{t}.\overrightarrow{A}} \vec{s}.B}\}.$$

   This operation represents a powerful way of constructing probabilistic databases from complete relations.

   Additionally, there is another operation introduced that uses repair-key (19): **power-world-set** operation. Let $A \ni sch(R)$, then

$$pws(R) := \pi_{sch(R)}\left(\sigma_{A=1}\left(repair - key_{sch(R)@P}\left(R \times \rho_A(\{0,1\}) \times \rho_P(\{1\})\right)\right)\right).$$

Each world is a subset of R, and the set of worlds created is called the *powerset* of R.

Example 7. Let us consider the example with tossing a coin. To introduce some different probabilities let us say it is a biased coin. For an experiment we will throw the coin twice. We start with a certain database:

R	$Toss$	$Face$	$FProb$
	1	H	0.4
	1	T	0.6
	2	H	0.4
	2	T	0.6

that represents the possible outcomes of tossing the coin twice. We turn this into a probabilistic database that represents this information using alternative possible worlds for the four outcomes using the query $S := repair - key_{Toss@FProb}(R)$:

U_R	$V \to D$	$Toss$	$Face$	$FProb$
	$1 \to H$	1	H	0.4
	$1 \to T$	1	T	0.6
	$2 \to H$	2	H	0.4
	$2 \to T$	2	T	0.6

W	V	D	P
	1	H	0.4
	1	T	0.6
	2	H	0.4
	2	T	0.6

The resulting possible worlds are:

S^1	$Toss$	$Face$	$FProb$
	1	H	0.4
	2	H	0.4

S^2	$Toss$	$Face$	$FProb$
	1	H	0.4
	2	T	0.6

S^3	$Toss$	$Face$	$FProb$
	1	T	0.6
	2	H	0.4

S^4	$Toss$	$Face$	$FProb$
	1	T	0.6
	2	T	0.6

with probabilities $p^1 = p * \frac{0.4}{0.4+0.6} * \frac{0.4}{0.4+0.6} = 0.16$, $p^2 = p^3 = 0.24$, and $p^4 = 0.36$.

4. **Choice-of** operation: introduces uncertainty like the repair-key operation, but can only cause a polynomial, rather than exponential, increase of the number of possible worlds (9).

   Semantics:

$$choice - of_{\vec{A}@B}(R) = \text{R} \bowtie \text{repair} - \text{key}_{\emptyset@\text{B}}(\pi_{\vec{A},B}(\text{R})).$$

   R must satisfy the functional dependency $R : \vec{A} \rightarrow B$ and the *B* values must be real numbers greater than 0.

   Example 8. For a probabilistic database R:

R^1	A	B	C
	a	1	c
	a	1	d
	b	3	e

$Pr = 0.5$ ($example\ of\ one\ of\ the\ worlds$)

Operation $choice - of_{\vec{A}@B}(R)$ produces the following worlds (for R^1):

$S^{1.1}$	A	B	C	
	a	1	c	
	a	1	d	$Pr = 0.5 * \frac{1}{4} = \frac{1}{8}$

$S^{1.2}$	A	B	C	
	b	3	e	$Pr = 0.5 * \frac{3}{4} = \frac{3}{8}$

5. **Assert** operation: takes a Boolean positive relational algebra query ϕ in SQL syntax as an argument, i.e., a select-from-where-union query without aggregation and conditions the database using this *constraint* ϕ. Conceptually, it removes all the possible worlds in which ϕ evaluates to false and renormalizes the probabilities so that they sum up to one again (9).

   Semantics:

$$[[assert(\varphi)]](W) := \left\{ \left(R_1, \ldots, R_k, \frac{p}{p_0} \right) \middle| (R_1, \ldots, R_k, p) \in W, \right.$$

$$\left. (R_1, \ldots, R_k) \models \varphi, p_0 = \sum_{(R'_1, \ldots, R'_k, p) \in W, (R'_1, \ldots, R'_k) \models \varphi} p \right\}.$$

   If the condition is such that the operation would delete all possible worlds when executed, it fails with an error (and leaves the database untouched). Example of the assert operation is presented in the following sections.

Representation model

Here we describe the method for representing and storing probabilistic data in MayBMS. Let us start with a motivating example.

Example 9. Network marketing for cosmetics and perfume companies involves a huge amount of manually filled in forms: the employees have to spend a lot of time "on the streets" collecting future clients' data, such as names, phone numbers and product preferences. The data in these forms afterwards has to be put into a database. It can be done automatically or manually, but neither the person doing the job or the program are able recognize handwriting 100% correctly. Therefore, uncertainty may be introduced into the database for some of the answers. *Figure 6* shows the example of a form and three filled in forms for a perfume selling company.

Perfumes campaign form

Name:

Mobile Number:

Preferable Perfumes Brands: (1) Gucci □ (2) Dior □

(3) Calvin Klein □ (4) Dolce & Gabbana □

Figure 6 (a)

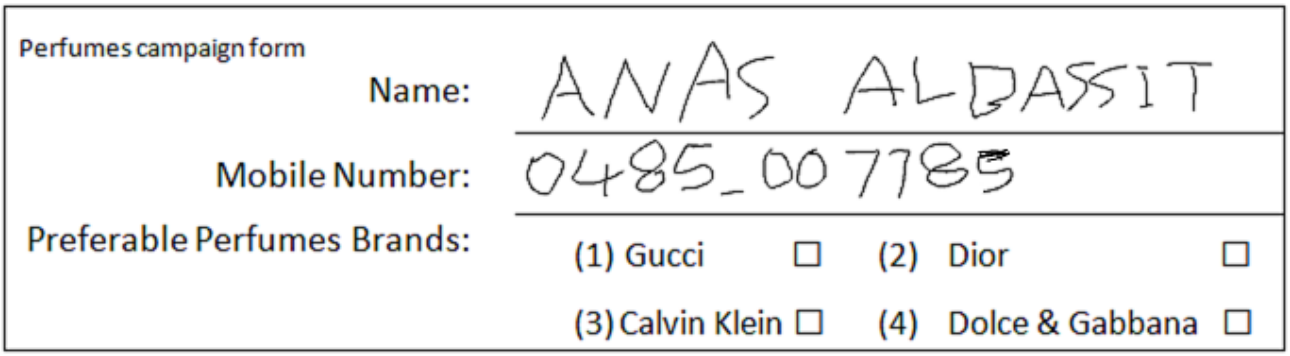

Figure 6 (b)

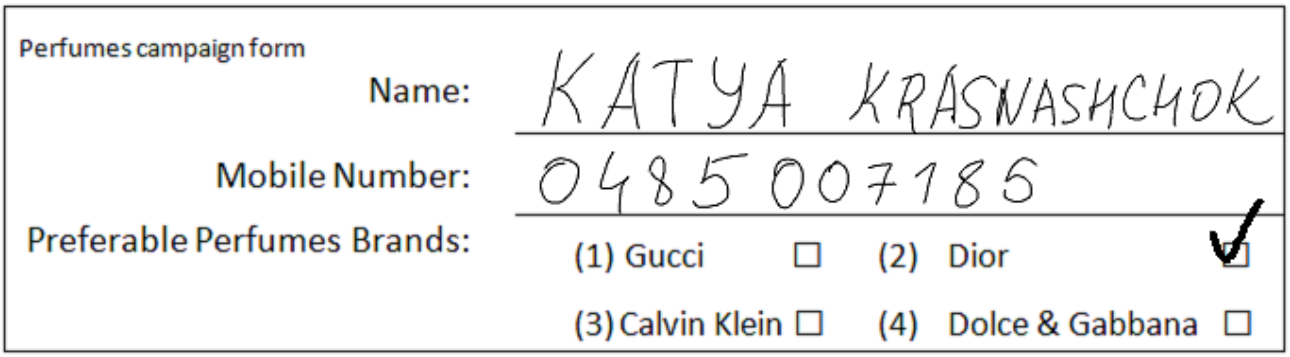

Figure 6 (c)

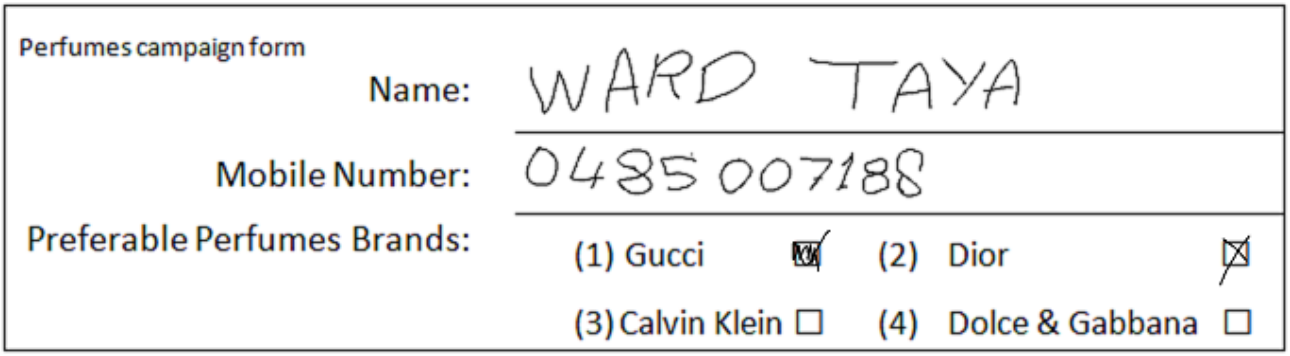

Figure 6 (d)

Each form consists of the name, phone number (for calling the potential client later) and preference in the brands of perfume. The first person, ANAS ALBASSIT, have not checked any perfume brands. The second person – WARD TAYA – checked "Gucci" at first, but then changed his choice to "Dior", however, it could also be the opposite. The phone numbers can also be interpreted differently (and we assume that there cannot be two people with the same phone numbers). Since all three people live in Brussels and are clients of the same mobile operator, all numbers starting with 0485 007, but the ending parts must vary. As can be seen from the *Figure 6*, ANAS' number could be ending on 185 or 785, KATYA's may either be 185 or 186, and WARD's number looks like both 186 and 188.

In an SQL database uncertainty can be managed using null values (9). But in this case, we lose information about the possible values. And obviously, it is not possible to express correlations, i.e. with null values we cannot ensure that two people will not have the same phone number. In our example, we can exclude the case that ANAS and KATYA both have the number ending on 185, also KATYA and WARD cannot have 186 at the same time. Finally, by using null values it is impossible to store probabilities for the possible worlds.

This example shows us three desired features of the representation system (2): expressiveness - the power to represent probabilistic databases, succinctness - space-efficient storage of the uncertain data, and efficient real-world query processing.

MayBMS uses a relational representation system called U-relational databases, which is based on probabilistic versions of the conditional tables (c-tables) mentioned in above sections. Conditional tables are a relational representation system that uses "labeled null values" or "variables", i.e. null values with a name. The name makes it possible to use the same variable x in several fields of a database, indicating that the value of x is unknown but must be the same in all those fields in which x occurs. Tables with variables are also known as v-tables (2).

C-tables are v-tables extended by a column that holds a local condition: each tuple of a c-table has a Boolean condition consisting of AND, OR, and NOT combinations of atomic conditions. Each atomic condition can be of two forms: $x = C$ or $x = y$, where C is a constant and x and y are variables.

Conditional tables are a so-called "strong representation system": the set of worlds defining the result of a query in each possible world that is a conditional table can again be represented by a conditional table (9).

In the model used by MayBMS, probabilistic databases are finite sets of possible worlds with probability weights attached to them. So each variable can be considered a finite random variable. Additionally, it is assumed without loss of generality that each atomic condition only compares a variable to a constant: $x = C$ (9).

If the initial c-table has each local condition as a conjunction of no more than k atomic conditions, then a query on this database will give us a c-table in which each local condition is a conjunction of no more than k´ conditions, where k´ only depends on k and the query (not on the data). If k is small, it is reasonable to actually hard-wire it in the schema, and represent local conditions by k pairs of columns storing conditions $x = C$. In the current implementation of the MayBMS system random variables are assumed independent (2).

Example 10. The tables in Figure 7 is a U-relational database for the marketing campaign from Example 9, probabilities added for the alternative values of different fields (table W).

$U_{R[MN]}$	$V \to D$	TID	MN
	$x \to 1$	t_1	048500185
	$x \to 2$	t_1	048500785
	$y \to 1$	t_2	048500185
	$y \to 2$	t_2	048500186
	$z \to 1$	t_3	048500186
	$z \to 2$	t_3	048500188

Figure 7 (a)

$U_{R[N]}$	TID	N
	t_1	ANAS ALBASSIT
	t_2	KATYA KRASNASHCHOK
	t_3	WARD TAYA

Figure 7 (b)

$U_{R[PPB]}$	$V \to D$	TID	PPB
	$u \to 1$	t_1	1
	$u \to 2$	t_1	2
	$u \to 3$	t_1	3
	$u \to 4$	t_1	4
	$w \to 1$	t_2	2
	$v \to 1$	t_3	1
	$v \to 2$	t_3	2

Figure 7 (c)

W	$V \rightarrow D$	P
	$x \rightarrow 1$	0.4
	$x \rightarrow 2$	0.6
	$y \rightarrow 1$	0.7
	$y \rightarrow 2$	0.3
	$z \rightarrow 1$	0.9
	$z \rightarrow 2$	0.1
	$u \rightarrow 1$	0.25
	$u \rightarrow 2$	0.25
	$u \rightarrow 3$	0.25
	$u \rightarrow 4$	0.25
	$w \rightarrow 1$	1
	$v \rightarrow 1$	0.2
	$v \rightarrow 2$	0.8

Figure 7 (d)

A U-relational database is constructed as a set of independent random variables with finite domains (in our example they are x, y, z, u, v and w), plus a set of U-relations, and a ternary table W ("the world-table") for representing distributions of probabilities. For each variable, the W table stores which values it can take and with what probability. The schema of each U-relation is a set of pairs (V_i, D_i) representing variable assignments plus a set of value columns for representing the data values of tuples (9).

U-relational databases are a complete representation system for (finite) probabilistic databases. Hence, this form can be used to represent any probabilistic database. Starting from an initial database, the result of the query on this database can again be represented as a U-relational database. It is implied that any correlation among tuples also can be represented the same way, despite the fact that currently the random variables constructing those correlations are considered independent (9).

Query language

The query language of MayBMS is based on SQL. In terms of U-relations traditional relational tables are a special case: they are called typed-certain (t-certain) tables. Tables that are not t-certain are called uncertain (2). Semantically, we can say that

$$cert(R) = \pi_{sch(R)}(\sigma_{P=1}(conf(R))).$$

In MayBMS, on t-certain tables full SQL is supported. For uncertain tables, there are some restrictions because of the specifics of the probabilistic databases. In particular, MayBMS does not support aggregates like sum or count on uncertain relations, the reason being that these aggregates will produce exponentially many different numerical results in the various possible worlds, and there is no way to efficiently represent these results. However, MayBMS has its own set of aggregations especially for uncertain tables, such as expected sums and counts - esum and ecount. Moreover, the conf operation is also an aggregate. All aggregates on uncertain tables produce t-certain tables (9).

The MayBMS query language is built from uncertain and t-certain queries. The uncertain queries are those that produce a possibly uncertain relation - a U-relation, and t-certain queries produce traditional database tables – t-certain tables.

The basic operations in MayBMS are:

1. **Repair-key**:

```
repair key <attributes> in
(<t-certain-query> | <t-certain-
relation>)
[ weight by <expression> ]
```

   The repair-key operation turns a t-certain-query (or a t-certain-relation) into the set of worlds consisting of all possible maximal repairs of key attributes. The <expression> is used for weighting the newly created alternative repairs. The value of <expression> cannot be negative. Without the weight by clause, a uniform probability distribution is assumed: if there are n tuples with the same key, each of them is associated with a probability 1/n. The tuples with the zero probability are not included in the resulting possible worlds. Repair-key can be placed wherever a select statement is allowed in SQL (note that repair-key is a query, rather than an update statement) (2).

2. **Pick-tuples**:

```
pick tuples from
<t-certain-query> | <t-certain-relation>
[independently]
[with probability <expression>];
```

   The pick-tuples selects a subset of the tuples of t-certain query or relation. For the <expression> only values in (0,1] are accepted. If <expression> is missing every tuple in a possible world has the probability 0.5. Tuples with resulting probability 0 are ignored. The operation can be placed wherever a select statement is allowed (2).

3. **Possible**:

```
select possible <attributes> from <query>
| <relation>;
```

   This operation selects the set of tuples that appear in at least one possible world. This construct is a shortcut for the query which selects all distinct tuples with confidence greater than zero.

4. **Confidence and approximate aggregates**:

Here is the summary of the aggregation functions introduced by MayBMS:

- **argmax:**

```
argmax(<argument-attribute>, <value-
attribute>)
```

outputs all the argument-attribute values in the current group (determined by the group-by) whose tuples have a maximum value-attribute value within the group. Can be used on all relations and queries (9).

- **conf:**

```
select <attribute | conf()> [, ...]
from <query> | <relation>
group by <attributes>;
```

computes for each possible distinct tuple that occurs in an uncertain relation in at least one possible world, the sum of the probabilities of the worlds in which it occurs. Can only be used on a uncertain query and the output is a t-certain relation.

- **tconf:**

```
select <attribute | tconf()> [, ...]
from <query> | <relation>;
```

for each possible tuple computes the sum of the probabilities of the worlds where it appears. It is different from conf: it does not eliminate duplicates. Can only be used on a t-uncertain query and the output is a t-certain relation.

- **aconf(ε, δ):**

```
select <attribute | aconf(<epsilon>,
<delta>)> [, ...]
from <query> | <relation>
group by <attributes>;
```

computes for each possible distinct tuples of the target list that occurs in at least one possible world, the approximate sum of the probabilities of the worlds in which it occurs. If p is the exact conf and ˆp is the approximate sum (aconf), the approximation has the property:

$$\Pr[|p - \hat{p}| \geq \varepsilon * p] \leq \delta.$$

- **esum** and **ecount:**

```
select <attribute | esum(<attribute>) |
ecount()> [, ...]
from <query> | <relation>
group by <attributes>;
```

these aggregates compute expected sums and counts across groups of tuples. Esum and ecount can only be used on a t-uncertain query and the output of the query is a t-certain relation (2).

5. **Updates.** MayBMS supports the usual schema alteration and update statements of SQL (2), such as:

   - **insertions** of the form

     ```
     insert into <uncertain-table>
     (<uncertain-query>);
     ```

   - **DDL operations** such as

     ```
     create table <uncertain-table> as
     (<uncertain-query>);
     ```

   - **deletions**

     ```
     delete from <uncertain-table> where
     <condition>;
     ```

6. **Conditioning operation assert:**

   Example 11. Consider the four possible worlds for the R[MN] (mobile number) relation of the perfume marketing campaign example.

R^1	TID	MN
	t_1	048500185
	t_2	048500185

R^2	TID	MN
	t_1	048500185
	t_2	048500186

R^3	TID	MN
	t_1	048500785
	t_2	048500185

R^4	TID	MN
	t_1	048500785
	t_2	048500186

To assert the functional dependency R: MN → TID = Q (Boolean) (i.e. no two people can have the same phone number), we execute assert(Q): the operation will delete the first world and renormalize the probabilities so they sum up to 1.

As we can see from this example, assert operation can apply a set of constraints to a probabilistic database and return cleaned, "more certain" database.

Case Study. “LinkedIn”- like social network

As we all know, IT4BI is more than a master's degree program. It gathers the best students the world over and put them together to make a group of elites and seize out the best of themselves.

This year two people of this group have met and added one each another on LinkedIn, those two are **Katya** and **Anas**. Since they are connected now on this social network, their old networks are connected now in a way or another. After calculating how likely one in Katya's network can be connected to one in Anas's network based on their skills, their previous or current jobs, their previous or current universities and schools or even the places they have been to in the same periods like conferences or meeting or workshops, we got the best 12 people from each one's network to be candidates to recommend them to one each another in the future. The full code of the case study can be found in the accompanying text file.

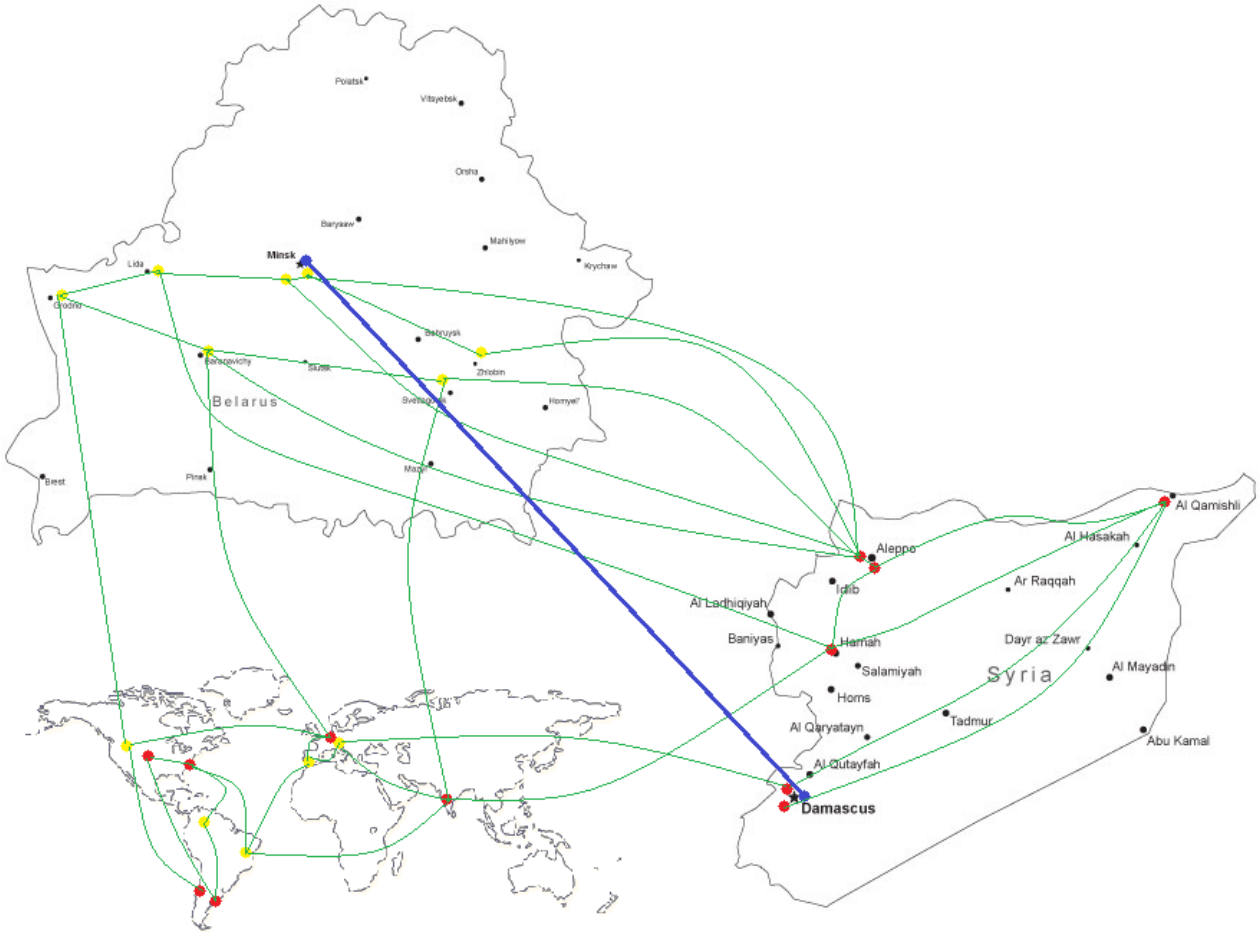

Figure 8

In addition, we can get these two groups (Figure 8) and make some experiments on them. First, as we realize easily most of the people in the two groups are from Syria or Belarus the countries which Anas and Katya came from, respectively. This gives us the chance to ask some questions like

1) What is the probability to find a Belarusian in this group connected to a Syrian, and vice versa?

We can as well ask more interesting questions like:

2) What is the probability to find three people connected in the resulted graph and they are from different nationalities?

Let us start with the first question and to answer this simple question we need to create a table of the people inside this graph

```
create table Person (ID integer, varchar name, nationality varchar);

insert into Person values (1, 'Sami','SY');
insert into Person values (2, 'Eias','SY');
insert into Person values (3, 'Alaa','SY');
insert into Person values (4, 'Ghazal','SY');
insert into Person values (5, 'Rami','SY');
insert into Person values (6, 'Jad','SY');
insert into Person values (7, 'Ricardo','CL');
insert into Person values (8, 'Nacho','AR');
insert into Person values (9, 'Bakari','IN');
insert into Person values (10, 'Jack','US');
insert into Person values (11, 'Samuel','US');
insert into Person values (12, 'Finn','DE');
insert into Person values (13, 'Tanya','BY');
insert into Person values (14, 'Elena','BY');
insert into Person values (15, 'Polina','BY');
insert into Person values (16, 'Olga','BY');
insert into Person values (17, 'Alex','BY');
insert into Person values (18, 'Vlad','BY');
insert into Person values (19, 'Sasha','US');
insert into Person values (20, 'Pavel','BR');
insert into Person values (21, 'Marik','ES');
insert into Person values (22, 'Viktor','VE');
insert into Person values (23, 'Anton','BY');
insert into Person values (24, 'Tyler','FR');
```

Then we need to create a table that defines the probability that connects each one to another and in this example we assume that we have a complete graph which means a table of C(24,23)*2 tuples. The number 2 means that for each relation between two people there's a probability that they are connected (p) and another one ($q = 1 - p$) that there not connected.

So we start with creating a table of probabilities called FOF and then fill it with the data with a condition($ID1 < ID2$).

```
create table FOF (ID1 integer, ID2 integer, bit integer, p real);

insert into FOF values (1, 2,1,0.02);
insert into FOF values (1, 3,1,0.12);
...
insert into FOF values (8, 17,1,0.07);
insert into FOF values (8, 18,1,0.07);
insert into FOF values (8, 19,1,0.07);
insert into FOF values (8, 20,1,0.07);
insert into FOF values (8, 21,1,0.07);
...
insert into FOF values (11, 15,1,0.33);
insert into FOF values (11, 16,1,0.33);
insert into FOF values (11, 17,1,0.33);
insert into FOF values (11, 18,1,0.33);
insert into FOF values (11, 19,1,0.33);
insert into FOF values (11, 20,1,0.23);
...
insert into FOF values (21, 24,1,0.04);
insert into FOF values (22, 23,1,0.04);
insert into FOF values (22, 24,1,0.04);
insert into FOF values (23, 24,1,0.04);
```

Then we insert the inverse of this table to get the full graph

```
insert into FOF select ID2 as ID1, ID1 as ID2, bit, p from FOF;
```

And finally insert the probability of not connecting $(q = 1 - p)$

```
insert into FOF select ID1, ID2, 0 as bit, 1-p as p from FOF;
```

Now we apply the function $'repair\ key'$ on $'ID1'$ and $'ID2'$ weighted by the probability $'P'$ to add to our information a probabilistic meaning

```
create table FOFP as repair key ID1,ID2 in FOF weight by p;
```

We delete then the probability of not being connected because we will not need this in the next queries so we raise the efficiency of the operations

```
delete from FOFP where bit=0;
```

After all, the resulted table will have the shape of

```
select id1,id2,p from FOFP;


  id1 |  id2 |   p  | _v0  | _d0  | _p0
------+------+------+------+------+------
   1  |   2  | 0.02 | 1824 | 4627 | 0.02
   1  |   3  | 0.12 | 1825 | 4611 | 0.12
   1  |   4  | 0.08 | 1826 | 4034 | 0.08
   1  |   5  | 0.13 | 1827 | 4709 | 0.13
   1  |   6  | 0.02 | 1828 | 3882 | 0.02
   1  |   7  | 0.02 | 1829 | 4597 | 0.02
   1  |   8  | 0.02 | 1830 | 4479 | 0.02
   1  |   9  | 0.02 | 1831 | 4573 | 0.02
   1  |  10  | 0.02 | 1832 | 4644 | 0.02
   1  |  11  | 0.03 | 1833 | 3977 | 0.03
   1  |  12  | 0.03 | 1834 | 4451 | 0.03
   1  |  13  | 0.03 | 1835 | 4412 | 0.03
   1  |  14  | 0.03 | 1836 | 3978 | 0.03
   1  |  15  | 0.03 | 1837 | 3873 | 0.03
   1  |  16  | 0.03 | 1838 | 3708 | 0.03
   1  |  17  | 0.03 | 1839 | 4510 | 0.03
   1  |  18  | 0.03 | 1840 | 4427 | 0.03
   1  |  19  | 0.03 | 1841 | 3947 | 0.03
   1  |  20  | 0.03 | 1842 | 4224 | 0.03
   1  |  21  | 0.03 | 1843 | 3715 | 0.03
   1  |  22  | 0.03 | 1844 | 3651 | 0.03
   1  |  23  | 0.03 | 1845 | 4160 | 0.03
 ...  ...
 ...  ...
  23  |  20  | 0.04 | 2349 | 4373 | 0.04
  24  |  20  | 0.04 | 2372 | 3846 | 0.04
  22  |  21  | 0.04 | 2327 | 4702 | 0.04
  23  |  21  | 0.04 | 2350 | 3632 | 0.04
  24  |  21  | 0.04 | 2373 | 4684 | 0.04
  23  |  22  | 0.04 | 2351 | 3757 | 0.04
  24  |  22  | 0.04 | 2374 | 4457 | 0.04
  24  |  23  | 0.04 | 2375 | 3918 | 0.04
(552 rows)
```

Now we start with the first query:

1- What is the probability to find a Belarusian in this group connected to at least one Syrian?

```
select conf() from Person p1,Person p2, FOFP a1, FOFP a2
where p1.ID=a1.ID1 and p2.ID=a2.ID1
        and a1.ID1=a2.ID2
        and p1.nationality='BY' and p2.nationality='SY';
```

And the result will be:

```
  conf
---------
 0.72466
(1 row)
```

2- What is the probability to find a Syrian in this group connected to at least one Belarusian?

```
select conf() from Person p1,Person p2, FOFP a1, FOFP a2
where p1.ID=a1.ID1 and p2.ID=a2.ID1
        and a1.ID1=a2.ID2
        and p1.nationality='SY' and p2.nationality='BY';
```

The result in this case:

```
  conf
---------
 0.652152
(1 row)
```

3- What is the probability to find three people connected in the resulted graph and they are from different nationalities?

```
select conf() from Person p1,Person p2 , Person p3,
                   FOFP a1, FOFP a2, FOFP a3
where p1.ID=a1.ID1 and p2.ID=a2.ID1 and p3.ID=a3.ID1 and
        a1.ID1=a2.ID2 and a2.ID1=a3.ID2 and a3.ID1=a1.ID2
        and p1.nationality<>p2.nationality
        and p2.nationality<>p3.nationality
        and p3.nationality<>p1.nationality;
```

And the result is:

```
  conf
----------
 0.531145
(1 row)
```

An even more efficient implementation uses the approximated confidence ($aconf(a,b)$) instead of the accurate one ($conf$) that deviates from the correct probability p by more than a, p is less than b. All this since the number of tuples is big and it needs much time to calculate it, despite that the approximated one gives almost the right answers in much less time.
Let us then apply $aconf(.05,.05)$

```
select aconf(.05,.05) from Person p1,Person p2 , Person p3,
                           FOFP a1, FOFP a2, FOFP a3
where p1.ID=a1.ID1 and p2.ID=a2.ID1 and p3.ID=a3.ID1 and
        a1.ID1=a2.ID2 and a2.ID1=a3.ID2 and a3.ID1=a1.ID2
        and p1.nationality<>p2.nationality
        and p2.nationality<>p3.nationality
        and p3.nationality<>p1.nationality;
```

And the result will be, in this case:

```
  aconf
----------
 0.532031
(1 row)
```

Now let us assume that Katya and Anas decided create a new startup and they decided to rely on one team of developers to help them. This team should be located in Syria or in Belarus so they can communicate easier in terms of geography and languages.
So they started by gathering the skills they need to establish the startup which are:

```
1, Java
2, PHP
3, Symfony
4, C#
5, MySQL
6, Android
7, IOS
8, AJAX
9, XML
10, MVC
```

And then they decided each on 5 of their friends

```
Katya's Belarusian Team:          Anas's Syrian Team:

1, Tanya                          1, Sami
2, Elena                          2, Eias
3, Polina                         3, Alaa
4, Olga                           4, Ghazal
5, Alex                           5, Rami
```

Again, we got the profiles of those people on LinkedIn and we collected their skills.
Now let us assume that they invited both teams, they want for sure (certain probability) to collect the skills that they can get from each team if we know that most probably one member of each suggested team will not be able to work with them, and based on that choose the more suitable team to represent the startup.
To get this information we start by creating the tables we need:

```
create table Skill (ID integer, skill varchar );

insert into Skill values (1, 'Java');
insert into Skill values (2, 'PHP');
insert into Skill values (3, 'Symfony');
insert into Skill values (4, 'C#');
insert into Skill values (5, 'MySQL');
insert into Skill values (6, 'Android');
insert into Skill values (7, 'IOS');
insert into Skill values (8, 'AJAX');
insert into Skill values (9, 'XML');
insert into Skill values (10, 'MVC');
```

And

```
create table Person_Skill (PID integer, SID integer);

insert into Person_Skill values (1, 1);
insert into Person_Skill values (1, 4);
insert into Person_Skill values (1, 5);
insert into Person_Skill values (1, 3);
insert into Person_Skill values (5, 5);
...
insert into Person_Skill values (13, 6);
insert into Person_Skill values (14, 7);
insert into Person_Skill values (15, 8);
...
insert into Person_Skill values (4, 7);
insert into Person_Skill values (3, 8);
insert into Person_Skill values (2, 9);
insert into Person_Skill values (13, 10);
```

Then we create a helper table that contains for each one of the selected people and their nationality;

```
Create table PP as select distinct P.ID,P.nationality
from Person_Skill PS, Person P
where PS.PID=P.ID;
```

We start theoretically by choosing the country which we want to represent us like this

$$U = choice\_of_{nationality}\ (\ PP\ )$$

That will distribute the data inside the table SS into two worlds each one contains the data in PS that is related to one of the countries (SY, BY).
Now let us assume that one of the people in each group leaves the team

$$V = \pi_{1.nationality,2.PID}(choice\_of_{PID}\ (U) \bowtie_{1.nationality=2.nationality \wedge 1.PID \neq 2.PID} PP$$

That will distributes the data again into a number of worlds And each one contains 4 people from different countries which means.

$$(5\ (one\ for\ each\ Syrian\ leaves) + 5\ (one\ for\ each\ Belarusian\ leaves) = 10\ worlds)$$

The first two relations could be represented in one query

```
create table RemainingPeople as
select PP.nationality, PP.ID
from PP,
(repair key dummy
in (select 1 as dummy, * from PP)) Choice
where PP.nationality = Choice.nationality
and PP.ID <> Choice.ID;
```

And the resulted table will be

```
 nationality | id | _v0  | _d0  | _p0
-------------+----+------+------+-----
 SY          |  1 | 2387 | 4942 | 0.1
 SY          |  1 | 2387 | 4941 | 0.1
 SY          |  1 | 2387 | 4943 | 0.1
 SY          |  1 | 2387 | 4940 | 0.1
 SY          |  2 | 2387 | 4942 | 0.1
...
 BY          | 16 | 2387 | 4946 | 0.1
 BY          | 17 | 2387 | 4944 | 0.1
 BY          | 17 | 2387 | 4947 | 0.1
 BY          | 17 | 2387 | 4949 | 0.1
 BY          | 17 | 2387 | 4946 | 0.1
(40 rows)
```

Where $\_d0$ represents the $world\ ID$ and hence it has 10 distinct values each one with a probability of 0.1.
Now let us compute the skills that we would gain in that case which are:

$$W = \pi_{nationality\ as\ team,skill}\ (\ V \bowtie PP \bowtie Person\_Skill\ )$$

And this is the query that represents this operation

```
create table SkillsAcquired as
select Q1.nationality as team, Q1.skill, p1, p2, p1/p2 as p
from (select R.nationality, S.skill, conf() as p1
      from RemainingPeople  R, Person_skill PS, Skill S
      where PS.SID=S.ID and R.ID = PS.PID
      group by R.nationality, S.skill) Q1,
      (select nationality, conf() as p2
      from RemainingPeople
      group by nationality) Q2
where Q1.nationality = Q2.nationality;
```

At this moment if we wanted to choose the skills that we obtain for certain we apply

$$C = Certain_{team,skill}\ (\ W\ )$$

```
select team, skill from SkillsAcquired where p=1;

team  |  skill
------+---------
 BY   | AJAX
 BY   | Android
 BY   | IOS
 BY   | Java
 BY   | MVC
 BY   | PHP
 BY   | Symfony
 BY   | XML
 SY   | AJAX
 SY   | Android
 SY   | C#
 SY   | IOS
 SY   | MVC
 SY   | MySQL
 SY   | XML
(15 rows)
```

Hence, we choose the first team to be our representative since we do not need to spend much money to acquire the incomplete skills list.
However, it is worth looking at the auxiliary table SkillAcquired:

```
 team |  skill   | p1  | p2  |  p
------+----------+-----+-----+-----
 BY   | AJAX     | 0.5 | 0.5 |   1
 BY   | Android  | 0.5 | 0.5 |   1
 BY   | IOS      | 0.5 | 0.5 |   1
 BY   | Java     | 0.5 | 0.5 |   1
 BY   | MVC      | 0.5 | 0.5 |   1
 BY   | PHP      | 0.5 | 0.5 |   1
 BY   | Symfony  | 0.5 | 0.5 |   1
 BY   | XML      | 0.5 | 0.5 |   1
 SY   | AJAX     | 0.5 | 0.5 |   1
 SY   | Android  | 0.5 | 0.5 |   1
 SY   | C#       | 0.5 | 0.5 |   1
 SY   | IOS      | 0.5 | 0.5 |   1
 SY   | Java     | 0.4 | 0.5 | 0.8
 SY   | MVC      | 0.5 | 0.5 |   1
 SY   | MySQL    | 0.5 | 0.5 |   1
 SY   | PHP      | 0.4 | 0.5 | 0.8
 SY   | Symfony  | 0.4 | 0.5 | 0.8
 SY   | XML      | 0.5 | 0.5 |   1
(18 rows)
```

This table contains the tuples $(x, y, p1, p2, p)$ such that:

- x is a team or a nationality;
- y is a skill;
- $p1$ is the probability that the chosen country is x and the skill y is gained;
- $p2$ is the probability that x is the chosen team;
- $p = p1/p2$ is the probability that skill y is gained if team x is chosen

Conclusion

A lot of applications can use imprecise data and therefore, probabilistic databases and the system for managing uncertain data. This brings the benefit of not having to clean the data, which can be a complicated, long and expensive process that still cannot guarantee a 100% correctness of the results. Moreover, probabilistic databases can be used successfully in data mining for finding correlations between various kinds of data, and in decision support and forecasting – for better predictions based on probabilities of different events.

But the difficulty of the problem of probabilistic inference resulted in the fact that currently there are no complete and efficient system exists for managing uncertain data. Various ongoing projects continuously try to achieve this goal in practice, supported by an increasing number of theoretical researches conducted in this area. Wide range of books, research papers and magazine articles has been written about probabilistic (or more generally, uncertain) data, each of them targeting different angles of this complex field: representation systems, query evaluation, user interface etc.

Most of those papers stick to the relational model for representing probabilistic data, but there are some that try to go beyond relational databases and introduce alternative ways of managing uncertain data, for example semistructured probabilistic databases (20) (which, to be completely fair, are XML extensions over the same relational databases).

Judging by the increasing interest in probabilistic data research, we can expect that in near future there may be new exciting discoveries in terms of solutions for the complexity and representation issues, which may not completely eliminate the problem, but will definitely (or, with a certain probability) lead to the development of new management systems, more efficient and ready to be deployed in large serious applications. And, consequently, real applications will bring more and more research possibilities.

In this report, we presented a state of the art overview of probabilistic databases, described various ways of data representation and executing queries. Also we provided a review of the probabilistic database management system we used for the case study – MayBMS – one of the most full, updated and documented systems out there, and, to our knowledge, having the most potential for development. In addition to that, we constructed a number of examples illustrating various theoretical concepts of probabilistic databases and possibilities of the chosen system: information extraction example, data cleaning (perfume marketing campaign) etc. Finally, we created a case study based on the social network like “LinkedIn” to show that probabilistic data can be successfully used in decision support and data analysis.

References

1. **Nilesh Dalvi, Christopher Ré, Dan Suciu.** *Probabilistic Databases: Diamonds in the Dirt (Extended version).*

2. **Aggarwal, Charu C.** *Managing and Mining Uncertain Data* . Boston/Dordrecht/London : Kluwer Academic Publishers.

3. **Koch, Christoph.** *MayBMS: A System for Managing Large Uncertain and Probabilistic Databases.* 2008/9.

4. **Pearl, Judaea.** *Probabilistic Reasoning in Intelligent Systems.* s.l. : Morgan Kaufmann, 1988.

5. **Wim Wiegerinck, Bert Kappen,Willem Burgers.** *Bayesian Networks for Expert Systems, Theory and Practical Applications.*

6. **Daphne Koller, Nir Friedman, Lise Getoor and Ben Taskar.** *Graphical Models in a Nutshell.*

7. **Amol Deshpande, Carlos Guestrin, Samuel R. Madden, Joseph M. Hellerstein, Wei Hong.** *Model-Driven Data Acquisition in Sensor Networks.* 2004.

8. **P. Andritsos, A. Fuxman, and R. J. Miller.** *Clean Answers Over Dirty Databases: a Probabilistic Approach.* 2006.

9. **Koch, Christoph.** *MayBMS: A System for Managing Large Uncertain and Probabilistic Databases.*

10. **Nilesh Dalvi, Dan Suciu.** *Management of Probabilistic Data: Foundations and Challenges.* 2007.

11. **Omar Benjelloun, Anish Das Sarma, Alon Halevy, Jennifer Widom.** *ULDBs: Databases with uncertainty and lineage.* 2006.

12. **L. Antova, C. Koch, and D. Olteanu.** *MayBMS: Managing incomplete information with probabilistic world-set decompositions (demonstration).* 2007.

13. **L. Antova, T. Jansen, C. Koch, and D. Olteanu.** *Fast and simple relational processing of uncertain data.* 2008.

14. **N. Friedman, L .Getoor, D. Koller, and A. Pfeffer.** *Learning probabilistic relational models.* 1999.

15. **N. Dalvi, Chris Re, and D. Suciu.** *Query Evaluation on Probabilistic Databases.* 2006.

16. **Dan Olteanu, Jiewen Huang, Christoph Koch.** *SPROUT: Lazy vs. Eager Query Plans for Tuple-Independent Probabilistic Databases.* 2009.

17. **Christopher Re, Dan Suciu.** *Materialized Views in Probabilistic Databases For Information Exchange and Query Optimization.* 2007.

18. **Suciu, Nilesh Dalvi and Dan.** *Efficient Query Evaluation on Probabilistic Databases.* 2007.

19. **Koch, Christoph.** *MayBMS: A System for Managing Large Uncertain and Probabilistic Databases (Talk).*

20. **Zhao, Wenzhong.** *Probabilistic Databases and Their Applications.* 2004.

Appendix A. The code for data cleaning example ("perfume marketing campaign")

```
--create initial table with initial data
create table PerfumeFormI (ID integer, TN integer, p float);
insert into PerfumeFormI values (1, 0485007185, .4);
insert into PerfumeFormI values (1, 0485007785, .6);
insert into PerfumeFormI values (2, 0485007185, .7);
insert into PerfumeFormI values (2, 0485007186, .3);
insert into PerfumeFormI values (3, 0485007186, .9);
insert into PerfumeFormI values (3, 0485007188, .1);

--op(repair_key)
create table PerfumeForm as
repair key ID in PerfumeFormI weight by p;

--take a look at the initial data
select ID, TN, conf() as initial
from PerfumeForm
group by ID, TN;

--create the table containing the violations and their propabilities
create table violations as
select S1.TN
from PerfumeForm S1, PerfumeForm S2
where S1.ID < S2.ID and S1.TN = S2.TN;

--create a table that contains how likely each person has each initial
  number
create table ID_TN_Final as
select Q1.ID, Q1.ID, p1, p2, p3, cast((p1-p2)/(1-p3) as real) as posterior
from( select ID, TN, conf() as p1
            from PerfumeForm
            group by ID, TN ) Q1,
      ((select TN, conf() as p2
            from violations
            group by TN)
      union
      ((select TN, 0 as p2 from PerfumeFormI)
      except
      (select possible TN, 0 as p2 from violations))) Q2,
      (select a.ID, conf() as p3
            from PerfumeForm a,violations b
            where a.TN=b.TN
            group by a.ID) Q3
where Q1.TN = Q2.TN and Q1.ID = Q3.ID;

--show the results
select * from ID_TN_Final;
```

Appendix B. Installing and running MayBMS system

Installing MayBMS

Installers for MayBMS are available for both Windows and Linux operating systems and can be downloaded at https://sourceforge.net/projects/maybms/. After you have obtained a copy of the installer, start it and follow the instructions.

Running MayBMS

After you have installed MayBMS, you can set up a database and start using it. Creating and accessing databases is the same as in PostgreSQL 8.3.3. Follow the links
http://www.postgresql.org/docs/8.3/interactive/tutorial-createdb.html
and
http://www.postgresql.org/docs/8.3/interactive/tutorial-accessdb.html.

Basically you have to go to the path where you installed MayBMS and consider running those instructions:

1- Create new database

Initdb -D mydbname

2- Start the database

Pg_cql start -D mydbname

3- Start new session

Psql template1

4- Stop the database

Pg_ctl stop -D mydbname