Introducing Mixed Tables

Lubomir Stanchev

Computer Science Department
California Polytechnic State University
San Luis Obispo, CA, USA
Email: lstanche@calpoly.edu

***Abstract*—Two approaches for extending relational database tables to allow storing uncertain and incomplete information have been proposed in the past. Grahne in 1984 introduced constraint tables that allow constraints to be attached to tuples and tables. Independently, Barbara et al. introduced in 1992 probabilistic tables that can contain random variables for some of the fields. In this paper, we combine the two approaches by introducing the concept of a *mixed table*: a table that allows storing both random variables and linear constraints on them.**

Keywords–c-tables; constraint databases; probabilistic databases; mixed tables.

I. INTRODUCTION

Most real-world information is incomplete or imprecise. For example, we may know that someone got good grades in algebra, but we may not know the precise grade. Similarly, we may know that a soldier is injured, but we may not know the extent of the injury. Or we may know that there is a 30% chance of rain tomorrow. Or maybe we know that there is a 90% probability that stock prices in the US will go down tomorrow if the Federal Reserve raises the key interest rate today. In this paper, we explore how such imprecise and probabilistic information can be represented in a tabular format without losing the richness of the data.

Many applications need access to incomplete information. For example, data mining algorithms can produce rules that have different levels of confidence. It is common for mobile sensors to produce conflicting information when operating in adverse weather conditions. Similarly, polling data is imprecise by nature. Storing such information in relational tables presents the advantage of allowing the use of exiting database technology, such as efficient querying, transaction control, logging and recovery, user authentication, and so on.

There is an obvious trade-off between the expressive power of the data representation and the complexity of query answering. For example, in the presence of Boolean constraints determining if a tuple belongs to every possible query result becomes as difficult as SAT [1], which is known to be NP-complete. Even in the absence of constraints, adding a random variable to every tuple that denotes the probability of the tuple existing (i.e., introducing *tuple-level uncertainty*) can make the problem of duplicate elimination over intermediate results #P-hard [2]. Fortunately, in most cases the high complexity is proportional only to the size of the incomplete information, which makes the algorithms practical when this size is limited. When this is not the case, Monte Carlo sampling algorithms that approximate the probability of a Boolean condition being true can be applied.

In this paper, we consider tables where random variables can occur for some of the fields. These are called probabilistic tables (or p-tables for short [3]). In addition, we allow constraints on these variables. Tables where constraints can be specified for some of the fields are called constraint tables (or c–tables for short [4]). We make the representation model even more expressive by considering bag semantics and allowing linear conditions over the random variables. This allows the tables to be closed under common relational algebra operations, such as projection, selection, join, duplicate elimination, and grouping and aggregation. We refer to such tables as *mixed tables* or *m-tables* for short. An example of an m-table is shown in Table I. The *global condition* field is part of every m-table. It specifies under what condition there will be tuples in the table. If the global condition is not satisfied, then the representation of the table will be empty. We assume that an empty condition is always true. A local condition is associated with each tuple. It specified the condition under which the tuple exists. In the example, either Bob or John goes to UCLA but not both of them because it is impossible that $x = 1$ and $x = 2$ at the same time. The variables y and x are random variable, where their distribution is shown in the lower part of Table I. For example, the random variable x shows that there is an equal probability that John or Bob studies in UCLA. Note that, as shown in Table I, every m-table can be represented using several relational tables (we will require that the random variables are initially independent and their distributions are discretized).

TABLE I. THE **Student** M-TABLE

name	*school*	*grade*	*local condition*
"John"	"UCLA"	y	$x = 1$
"Bob"	"UCLA"	"A"	$x = 2$

global condition:

value of y	*prob.*
"A"	0.6
"B"	0.3
"C"	0.1

value of x	*prob.*
1	0.5
2	0.5

In 1992, Rina Dechter wrote a survey paper on *constraint networks* [5] that shows how graph networks can be used to find the solution to a set of constraints. In 1985, Judea Pearl wrote a paper that introduces the concept of a *Bayesian network* for random variables [6]. The two concepts have been studies separately until Mateescu and Dechter introduced the concept of a *mixed network* [7]. This is a network that allows the specification of both constraints and dependencies between random variables. Here, we adopt the approach of the last paper and study how tables with linear constraints and random

 ISBN: 978-1-61208-472-5

variables can be stored and queried. While c-tables with linear conditions [1] and p-tables [3] have been extensively studded, we are not aware of any research that combines the two concepts in the presence of linear conditions. The advantage of our approach is that most relational algebra operations are straightforward to perform and have good running times. One drawback of our approach is that some of the complexity is buried in the data representation.

In this paper, we define the precise semantics of an m-table as a set of relational tables. We then examine the problem of m-table simplification and deciding when two m-tables are equivalent, that is, when they represent the same set of tables. As part of this study, we show why it is impossible to create a canonical form for an m-table. We also define different relational algebra operations, such as projection, selection, inner join, union, monus, and duplicate elimination. For each operation, we present the formal semantics, show why this semantics is well justified, and show an algorithm for performing the operation. The basic idea is that performing a relational algebra operation on one or more m-tables should result in an m-table that represents the set of tables that we will get if we performed the relational algebra operation on the tables that are represented be the input m-tables. When this is the case, we will say that the relational algebra operation is *sound and complete*. In this paper, we closely follow the research that was presented in [8]. The novelty is that now we allow random variables to be part of a table. We also optimize some of the algorithms and elaborate on how the duplicate elimination operation can be performed.

In what follows, in Section II we present related research. Section III shows the formal semantics of m-tables and defines what does it mean for two m-tables to be equivalent. Section IV presents our algorithms for performing the different relational algebra operations on m-tables. Lastly, Section V summarizes the paper and highlights avenues for future studies.

II. Related Research

We start by presenting relevant research in the area of incomplete databases. It turns out that the problem of representing incomplete information is as old as the relational model itself [9], [10], [11], [12], [13]. Imielinski and Lipsi [14] were among the first to propose richer semantics for incomplete information. Before that, the only option was to put "null" when the value of a field of a tuple is unknown. Later, Libkin and Wong [15] extended the research to tuples with bag semantics. Grahne extensively studied the problem of representing incomplete information [4], while Reiter [16] and Yuan et al. [17] explored algorithms for querying tables with null values. Libkin [18] addressed the problem of querying incomplete databases, while Buneman et. all [19] showed how a table can represent one of several possibilities. The most expressive representation of incomplete information that we are aware of is [8]. It presents a system that supports bag semantics with grouping and aggregation.

We next turn our attention to papers on databases with statistical data. It turns out that probabilistic databases are also as old as the relational model [20], [21], [22], [23]. Barbara et al. [24] were the first to show how random variables can appear inside tables and how the different relational algebra operations can be performed on such tables. Ge at al. [25] show a general approach to storing and querying probabilistic objects that can contain any number of attributes, while Suciu et al. [2] show a comprehensive overview of the current state-of-the-art in probabilistic databases. In particular, Jampani et al. [26] show how to apply a Monte Carlo algorithm to approximate the distribution of the variables in the result of applying different relational operations, while other papers [27], [28], [29], [30], [31] show how these probabilities can be exactly computed. Note that, unlike most papers that use exclusively the tuple level uncertainty approach, we combine the tuple and attribute level uncertainty approaches. Although this approach increases the complexity of our data representation model, it allows for more compact representation of information. As a final remark, note that we allow linear constraints to be associated with tuples. This is a generalization of the approach of previous papers that allow lineage to be associated with each tuple [2].

III. Introducing M-tables

In this section, we present the syntax and semantics of a mixed table (a.k.a. m-table) and discuss the questions of m-table simplification, m-table equivalence, and the existence of a canonical form for m-tables.

Definition 3.1 (syntax of an m-table): An *m-table* contains three parts: a bag ot *m-tuples*, a single *global condition*, and the distribution of the random variables. An m-tuple t with attributes $\{A_i\}_{i=1}^{a}$ is a sequence of mappings from A_i to $D(A_i) \cup \mathbf{V}_i$ (called the *main part* and denoted as $main(t)$) plus a *local condition* (denoted as $lc(t)$), where i ranges from 1 to a. $D(A_i)$ is used to represent the domain of the attribute A_i, while $\mathbf{V_i}$ is a set of random variables over $D(A_i)$. We will allow local and global conditions over the system $\langle \mathbb{R}, \{>, =, +\}\rangle \cup \langle \mathbb{S}, \{=, \neq\}\rangle$, where $\mathbb{R}$ is the set of real numbers and $\mathbb{S}$ is the set of all strings. We will use $gc(T)$ to denote the global condition of the m-table T. Note that we require that the distribution of the random variables be discretized so that it can be stored in a tabular format.

We will follow the approach of Imielinski and Lipski [32] and define the *rep* function. It shows the set of relational tables that an m-table represents. The novelty is that we will also add a probability to each relational table that shows how likely it is that the particular table contains the correct data. We can think of each relational table as the interpretation of the m-table (or its random variables) under some possible world, where the sum of the probabilities over all possible worlds is equal to one. Note that, unlike [32], we adopt the closed world assumption. That is, we assume that tuples that are not part of a table do not belong to the table. We also apply bag semantics, that is, we allow duplicate tuples.

Definition 3.2 (semantics of an m-table): An m-table T represents the following set of relational tables and associated probabilities.

$$rep(T) = \{\langle v(T), p(v)\rangle : v \text{ is such that } p(v) > 0\} \quad (1)$$

The function v interprets the variables in T as constants in the corresponding domains. The expression $p(v)$ denote the probability that the mapping v occurs, where this probability can be computed from the probability distribution of the random variables. If the probability distribution of a random variable is missing, then we assume uniform distribution. We next define the function v for m-tuples.

$$v(t) = \begin{cases} v(main(t)) & \text{if } v(lc(t)) \wedge v(gc(T)) \\ \varnothing & \text{otherwise} \end{cases} \quad (2)$$

In the above formula, t is an m-tuple of the m-table T. The formula $v(main(t))$ means that we apply the interpretation v to all the variables in the main part of the m-tuple t. The function v is generalized to m-tables as follows, where $\{| \cdot |\}$ is used to denote a bag of elements.

$$v(T) = \{|\ v(t) : t \text{ is such that } t \in T \wedge v(t) \neq \varnothing\ |\} \quad (3)$$

The above definition differs from the definition in [8] because now a probability is associated with each of the relational tables that are represented by an m-table. While it is certainly possible to extend the definition and define ordering over m-tables, we leave this topic as an area for future research.

For completeness, we show the four possible representations of our example **Student** m-table in Table II. For example, there is a 15% probability that John is enrolled in UCLA and has a grade of "B" because there is a 50% probability that he is enrolled in UCLA and a 30% probability that his grade is a "B", where the two probabilities are independent.

TABLE II. THE RESULT OF APPLYING THE *rep* FUNCTION TO THE **Student** TABLE

⟨ *name* "John"	*school* "UCLA"	*grade* "A" , 0.3⟩	⟨ *name* "John"	*school* "UCLA"	*grade* "B" , 0.15 ⟩
⟨ *name* "John"	*school* "UCLA"	*grade* "C" , 0.05 ⟩	⟨ *name* "Bob"	*school* "UCLA"	*grade* "A" , 0.5 ⟩

Note that our definition of an m-table differs from a common approach in probabilistic databases where a probability of existence is assigned to each tuple (i.e., tuple level uncertainty [2]). However, we can compute the probability of existence of a tuple t by computing the value of $p(lc(t) \wedge gc(t))$. The function p computes the probability that is associated with a linear expression. The function can be computed using a mixed probability network, where Mateescu et al. propose both a precise algorithm and approximate Monte Carlo sampling algorithm for computing the probability [7]. We can apply this algorithm because the distribution of the random variables is discretized. The precise algorithm creates a graphical model where a node is created for every random variable and random variables that are correlated (i.e., are part of the same linear constraint) are connected with edges. It then examines how the distribution of one random variable affects the distribution of the connected in the graph random variables. The sampling algorithm approximates the probability of a linear condition being true by simply generating random interpretations according to the probability distribution of the random variables and calculating the percent of time that the linear condition is true. The Monte Carlo algorithm should be applied whenever it is unfeasible to apply the precise algorithm.

A. M-table Simplification and Equivalence

We start this subsection with an algorithm that simplifies a linear condition. The *simplify* algorithm is presented in Figure 1. It can be used to simplify the local conditions and global condition of an m-tuple. The algorithm converts the condition in disjunctive normal form and then normalizes each conjunction using the *normalize* algorithm from [33]. The algorithm has the property that it will convert each of the conjunctions into a canonical form. However, as we have shown in [1], a canonical form for a general linear constraint does not exist. Informally, the reason is that there are different ways to describe a linear point set as the union of polyhedras. The algorithm takes exponential time relative to the size of the condition, which is not necessarily a big concern because in practice most conditions are relatively small. Depending on the performance requirments, the algorithm can use the exact or approximate algorithm for calculating the p function.

Algorithm 1 *simplify*(θ)

1: **if** $p(\theta) = 0$ **then**
2: **return false**
3: **end if**
4: **if** $p(\theta) = 1$ **then**
5: **return true**
6: **end if**
7: $\theta_1 \vee \ldots \vee \theta_n \leftarrow \theta$, where $\{\theta_i\}_{i=1}^{n}$ are conjunctions
8: **for** $i \leftarrow 1$ to n **do**
9: $\theta_i \leftarrow normalize(\theta_i)$
10: **end for**
11: **for** $i \leftarrow 1$ to n **do**
12: **if** $\theta_i \equiv$ **true** or $p(\theta_i) = 1$ **then**
13: **return true**
14: **end if**
15: **end for**
16: $\theta \leftarrow$**false**
17: **for** $i \leftarrow 1$ to n **do**
18: **if** $\theta_i \not\equiv$ **false** and $p(\theta_i) > 0$ **then**
19: $\theta \leftarrow \theta \vee \theta_i$
20: **end if**
21: **end for**
22: **return** θ

Figure 1. The algorithm for simplifying a linear condition.

Since a canonical form for m-tables does not exist, we can only simplify an m-table to make it more compact. The simplification algorithm is shown in Figure 2, where its main property is its correctness. That is, it does not change the set of relational tables that the input m-table represents.

The algorithm is identical to the algorithm from [1], where the only difference is that the *simplify* function considers the distribution of the random variables. The algorithm unifies tuples that are unifiable, which compacts the input m-table. Two m-tuples are unifiable when we know that exactly one of them can appear in any representation. For example, the two tuples in the example **Student** table are unifiable. Formal definition follows.

Definition 3.3 (m-tuple unification): The m-tuples t_1 and t_2 of the m-table T are unifiable exactly when the expression $lc(t_1) \wedge lc(t_2) \wedge gc(T)$ is not satisfiable.

The m-table simplification algorithm needs to consider all pairs of m-tuples and therefore will run in quadratic time relative to the size of the m-table. An example of applying the

 ISBN: 978-1-61208-472-5

Algorithm 2 *simplify*(T)

if $simplify(gc(T)) \equiv$ **false** or $p(gc(T)) = 0$ **then**
  return empty table
end if
for all $t \in T$ **do**
  if $simplify(lc(t) \wedge gc(t)) \equiv$ **false then**
    remove t from T
  end if
end for
while $\exists\{t_1, t_2\}$, s.t. $unifiable(t_1, t_2)$ **do**
  remove t_1 and t_2 from T
  crate a new m-tuple t
  $\mathbf{X} \leftarrow \{x_1, \ldots x_n\}$, a set of new variables
  $main(t) \leftarrow \{x_1, \ldots, x_n\}$
  $lc(t) \leftarrow (\mathbf{X} = main(t_1) \wedge lc(t_1)) \vee (\mathbf{X} = main(t_2) \wedge lc(t_2))$
  add t to T
end while
for all $t \in T$ **do**
  $lc(t) \leftarrow simplify(lc(t) \wedge gc(T))$
  if $lc(t) \equiv$ **false then**
    remove t from T
  else
    while $main(t)$ contains a variable x and $simplify(lc(t) \Rightarrow (x = c)) \equiv$ **true do**
      replace x with the constant c in $main(t)$
    end while
  end if
end for
$gc(T) \leftarrow$ empty
return T

Figure 2. The algorithm for simplifying a mixed table.

simplify function to the **Student** table is shown in Table III. Note that the m-table can be further simplified by removing the random variable x and the conditions $x = 1$ and $x = 2$. However, this involves a more evolved linear expression simplification algorithm, such as the one presented in [8].

TABLE III. THE RESULT OF *simplify*(**Student**)

name	*school*	*grade*	*local condition*
n	"UCLA"	g	$(x = 1 \wedge n =$ "$John$" $\wedge\, g = y) \vee$ $(x = 2 \wedge n =$ "Bob" $\wedge\, g =$ "A"$)$

value of y	*prob.*
"A"	0.6
"B"	0.3
"C"	0.1

value of x	*prob.*
1	0.5
2	0.5

We will say that two m-tables are equivalent when they represent the same set of relational tables. A formal definition follows.

Definition 3.4 (m-table equivalence): Two m-tables T_1 and T_2 are equivalent when $rep(T_1) \equiv rep(T_2)$. We will write $T1 \simeq T_2$ to denote that two m-tables are equivalent.

The following theorem shows that the m-table simplification algorithm does not change the meaning of an m-table.

Theorem 3.1: For any m-table T, $T \simeq simplify(T)$.

Lastly, the following theorem shows a practical way to check for the equivalence of two m-tables.

Theorem 3.2: $T_1 \simeq T_2$ exactly when $simplify(T_1 - T_2) = \varnothing$ and $simplify(T_2 - T_1) = \varnothing$, where "$-$" is the monus relational algebra operation, which is defined in the next section.

IV. RELATIONAL ALGEBRA OPERATIONS

It is expected that the different relational algebra operations will have "nice" properties. In particular, we want the result of applying a relational algebra operation on one or more m-tables to be an m-table. We also expect the result of the operation to be consistent with the semantic of the input m-tables (see Definition 3.2). Precise definition of these properties follows. Note that we will say that the tables $\{T_i\}_{i=1}^n$ are *allowable* for the relational algebra operation q when $q(\{T_i\}_{i=1}^n)$ is well defined. For example, union is allowed only on tables that have identical attributes.

Definition 4.1 (closed RA operation): A relational algebra operation q with arity n is closed if an only if the result of applying q to any allowable m-tables $\{T_i\}_{i=1}^n$ is an m-table, that is, $q(T_1, \ldots, T_n)$ is always an m-table.

Definition 4.2 (sound RA operation): A relational algebra operation q is sound when it produces only correct answers. Formally, $rep(q(T_1, \ldots, T_n)) \subseteq q(rep(T_1, \ldots, T_n))$ for any allowable tables $\{T_i\}_{i=1}^n$.

Note that we will some times abuse notation and use $q(rep(T_1, \ldots, T_k))$ to define the result of applying the operation q to each table in the set $\{rep(T_i)\}_{i=1}^k$.

Definition 4.3 (complete RA operation): A relational algebra operation q is complete exactly when all correct answers appear in the result, that is $q(rep(T_1, \ldots, T_n)) \subseteq rep(q(T_1, \ldots, T_n))$ for any allowable m-tables $\{T_i\}_{i=1}^n$.

In order for a relational algebra to be *well defined*, it is required that all operations are closed, sound, and complete. We next define relational algebra over m-tables that is well defined, where the definition of completeness for the monus operation will have to be slightly adjusted.

A. Projection

In this subsection, we will define duplicate preserving projection, which we will denote as π^d. The more common duplicate eliminating projection can be achieved by applying the duplicate elimination operation to the result.

The algorithm is shown Figure 3, where the algorithm simply selects the desired fields. The following theorem is not hard to prove and it is based on a similar theorem in [1].

Theorem 4.1: The projection operation is well-defined and the result can be computed in time that is linear to the size of the input table.

Table IV shows the result of applying the duplicate preserving projection operation on the *name* and *school* fields of the example **Student** table. Note that the local condition field does not need to be included in the list of projected fields because it cannot be projected out.

 ISBN: 978-1-61208-472-5

Algorithm 3 Evaluating $\pi^d_{\mathbf{A}}(T)$

```
1: S ← T
2: for all A ∈ attr(S) and A ∉ A do
3:    if x is a random variable that appears only in A then
4:       remove the probability table for x
5:    end if
6: end for
7: for all t ∈ S do
8:    remove attributes outside the set A
9: end for
10: return S
```

Figure 3. The algorithm for computing the projection over a mixed table.

TABLE IV. THE RESULT OF $\pi^d_{name,school}(\mathbf{Student})$

name	*school*	*local condition*
"John"	"UCLA"	$x = 1$
"Bob"	"UCLA"	$x = 2$

global condition:

value of x	*prob.*
1	0.5
2	0.5

B. Selection

Figure 4 shows how to apply the selection relational algebra operation to an m-table. It first replaces variables with constants where appropriate inside the main body of the table. It then adds the selection condition to all the local conditions. As a last step, the local conditions are simplified. Tuples that have a local condition that is false are removed. Similarly, tuples that have a local condition that is a tautology are left with empty local conditions. The following theorem is not hard to prove and it is based on a similar theorem in [1].

Theorem 4.2: The selection operation is well-defined and the result can be computed in time that is linear to the size of the input table plus the time to simplify the new local conditions.

Table V shows the result of applying $\sigma_{grade="A"}(\mathbf{Student})$. The algorithm first substitutes the value for the attribute *grade* of the first tuple with "A" because we only keep tuples if the grade is "A". Then the selection condition is added to the local conditions. Note that the condition that is added to the second tuple is "A"="A", which is a tautology and is therefore removed by the *simplify* algorithm.

TABLE V. RESULT OF $\sigma_{grade="A"}(\mathbf{Student})$

name	*school*	*grade*	*l. condition*
"John"	"UCLA"	"A"	$y =$ "A"$\wedge x = 1$
"Bob"	"UCLA"	"A"	$x = 2$

g.condition:

value of y	*prob.*
"A"	0.6
"B"	0.3
"C"	0.1

value of x	*prob.*
1	0.5
2	0.5

Algorithm 4 Evaluating $\sigma_\theta(T)$

```
1: S ← T
2: for all A_i such that θ has the form (A_i = value ∧ ...)
   and t.A_i = p do
3:    t[A_i] ←value
4:    if p does not appear anywhere else then
5:       remove the probability table for p
6:    end if
7: end for
8: for all t ∈ S do
9:    ψ(t) ← a substitution that substitutes every variable A_i
      with t[A_i] (the value for the attribute A_i in t)
10:   lc(t) ← simplify(lc(t) ∧ θ_ψ(t))
11:   if lc(t) ≡ true then
12:      make local condition of t empty
13:   end if
14:   if lc(t) ≡ false then
15:      remove t from S
16:   end if
17: end for
18: return S
```

Figure 4. The algorithm for computing selection over mixed table.

C. Natural Join

The algorithm for natural join is shown in Figure 5, where the algorithms for theta join and outer join are similar.

Algorithm 5 Evaluating $T_1(\mathbf{A},\mathbf{B}) \bowtie T_2(\mathbf{B},\mathbf{C})$

```
1: T ← empty m-table with the attributes of A ∪ B ∪ C
2: for all t_1 ∈ T_1 do
3:    for all t_2 ∈ T_2 do
4:       if simplify(t_1[B] = t_2[B]) ≢ false then
5:          t ← new tuple with attributes of T
6:          main(t) ← ⟨t_1, π^d_C(t_2)⟩
7:          lc(t) ← simplify(lc(t_1) ∧ lc(t_2) ∧ t_1[B] = t_2[B])
8:          add t to T
9:       end if
10:   end for
11: end for
12: gc(T) ← gc(T_1) ∧ gc(T_2)
13: return T
```

Figure 5. The algorithm for natural join of mixed tables.

The above algorithm simply joins tuples from the two m-tables that are *joinable*. Two tuples are joinable when the local condition of the new tuple is satisfiable. When two tuples are joined, the new local condition is the conjunction of the join condition and the local conditions of the two tuples. As a final step, the global conditions of the two tables are merged using conjunction. The following theorem is not hard to prove and it is based on a similar theorem in [1].

Theorem 4.3: The natural join operation is well-defined and the result can be computed in time that is proportional

to multiplying the sizes of the two tables times the time to perofrm the new linear condition simplification.

As is the case in relational databases, a join can be optimized using indexes. We leave the details as a topic for future research. Consider the **University** mixed table in Table VI. Table VII shows the result of the natural join of the **Student** and **University** tables. Note that the second tuple of the **University** table cannot join with any of the tuples in the **Student** table. Conversely, the first tuple of the **University** table joins with the two tuples in the **Student** table when $z = \text{“}UCLA\text{”}$. In the tuple-level uncertainty model, one can compute that the *marginal probability* of the first tuple in the result (i.e., the probability of the tuple existing in a representation) is $0.5 * 0.6 = 0.3$. However, our model does not explicitly store the marginal probabilities of the resulting tuples.

TABLE VI. THE **University** M-TABLE

school	*local condition*
z	
Cal Poly	

global condition:

value of z	*prob.*
“UCLA”	0.6
“Univeristy of California Los Angeles”	0.4

TABLE VII. THE RESULT OF **Student** $\bowtie$ **University**

name	*school*	*grade*	*local condition*
“John”	“UCLA”	y	$x = 1 \wedge z = \text{“}UCLA\text{”}$
“Bob”	“UCLA”	“A”	$x = 2 \wedge z = \text{“}UCLA\text{”}$

global condition:

value of y	*prob.*
“A”	0.6
“B”	0.3
“C”	0.1

value of x	*prob.*
1	0.5
2	0.5

value of z	*prob.*
“UCLA”	0.6
“Univeristy of California Los Angeles”	0.4

D. Union

Performing the duplicate preserving union of two m-tables is straightforward. Figure 6 shows the algorithm that simply merges the tuples of the two tables. The new global condition is the conjunction of the global conditions of the input tables.

Algorithm 6 Evaluating $T_1 \cup^d T_2$

1: $T \leftarrow$ empty m-table that has the attributes of T_1
2: **for all** $t_1 \in T_1$ **do**
3: add t_1 to T
4: **end for**
5: **for all** $t_2 \in T_2$ **do**
6: add t_2 to T
7: **end for**
8: $gc(T) \leftarrow gc(T_1) \wedge gc(T_2)$
9: **return** T

Figure 6. The algorithm for duplicate-preserving union of mixed tables.

The following theorem is not hard to prove and it is based on a similar theorem in [1]. Note that the m-table simplification algorithm can be applied to the resulting table in order to make it more compact, but this is not a requirement.

Theorem 4.4: The duplicate preserving union operation is well-defined and the result can be computed in time that is linear in the size of the input tables.

As an example, Table VIII shows the result of applying the duplicate preserving union on two copies of the *Student* table. The new table denotes that either we have two Bobs, or we have two Johns in UCLA, but not both.

TABLE VIII. THE RESULT OF **Student** $\cup^d$ **Student**

name	*school*	*grade*	*local condition*
“John”	“UCLA”	y	$x = 1$
“Bob”	“UCLA”	“A”	$x = 2$
“John”	“UCLA”	y	$x = 1$
“Bob”	“UCLA”	“A”	$x = 2$

global condition:

value of y	*prob.*
“A”	0.6
“B”	0.3
“C”	0.1

value of x	*prob.*
1	0.5
2	0.5

E. Monus

The duplicate-preserving monus operation is defined as follows $T_1 -^d T_2 = \{|t_{[k]} \mid t \in T_1 \wedge k = max(0, card(t, T_1) - card(t, T_2))|\}$. The *card* function returns the cardinality (a.k.a. number of appearances) of the tuple in the table, while $t_{[k]}$ denotes that the tuple t appears k times in the result. As expected, $t_{[0]}$ means that the tuple does not appear in the result.

We will define the algorithm that performs the monus operation using two two-dimensional arrays. Let $X[i, j]$ be the condition that must be true for the i^{th} tuple in T_1 to delete the j^{th} tuple in T_2. This condition is true under an interpretation that makes the main parts of the tuples the same and makes the local conditions of the two tuples and global conditions of the two tables true. Let $Y[i][j]$ be equal to 1 when the i^{th} in T_1 is deleted by the j^{th} tuple in T_2 and be equal to 0 otherwise. We will use Y to enforce the restriction that every tuple in T_2 can delete at most one tuple in T_1. Figure 7 shows the algorithm for performing the monus operation. The algorithm is an optimized version of the algorithm from [8]. Note that we do not specify the probability distributions of the two two-dimensional arrays of random variables and therefore uniform distribution is assumed.

The algorithm first removes tuples from T_2 that cannot delete tuples from T_1. It next copies tuples from T_1 that cannot be deleted by tuples from T_2 to the resulting set. As a final step, all the remaining tuples from T_1 are added to the result. For each of these tuples, a local condition is added that they will be part of the result only if they are not deleted by one of the tuples in T_2, where an interpretation of Y fixes which tuples in T_1 are deleted by which tuples in T_2.

The following theorem is not hard to prove and it is based on a similar theorem in [1].

Theorem 4.5: The monus operation is closed, sound, and it supports a version of completeness. Specifically, we need to modify the completeness condition as follows: $(Rep(T_1) - Rep(T_2)) \cup \{\varnothing\} \subseteq Rep(T_1 - T_2)$. The complexity of the operation is linear relative to the product of the sizes of the two tables plus the time to perform the new local condition simplification.

Algorithm 7 Evaluating $T_1 -^d T_2$

$V_1 \leftarrow T_1$
$V_2 \leftarrow T_2$
$R \leftarrow$ empty table that has the attributes of T_1
for all $t_1 \in V_1$ **do**
  if there is no tuple t_2 in V_2 such that $simplify(main(t_1) = main(t_2) \wedge lc(t_1) \wedge gc(t_1) \wedge lc(t_2) \wedge gc(t_2))$ **then**
    add t_1 to R
    remove t_1 from V_1
  end if
end for
for all $t_2 \in V_2$ **do**
  if there is no tuple t_1 in V_1 such that $simplify(main(t_1) = main(t_2) \wedge lc(t_1) \wedge gc(t_1) \wedge lc(t_2) \wedge gc(t_2))$ **then**
    remove t_2 from V_2
  end if
end for
$i \leftarrow 0$
for all $t_1 \in T_1$ **do**
  $j \leftarrow 0$
  for all $t_2 \in T_2$ **do**
    $X[i][j] \leftarrow simplify(main(t_1) = main(t_2) \wedge lc(t_1) \wedge gc(t_1) \wedge lc(t_2) \wedge gc(t_2))$
    $j \leftarrow j + 1$
  end for
  $i \leftarrow i + 1$
end for
$n \leftarrow$ number of tuples in V_1
$m \leftarrow$ number of tuples in V_2
$i \leftarrow 0$
for all $t \in V_1$ **do**
  $lc(t) \leftarrow lc(t) \wedge \neg(\bigvee_{j=1}^{m} (X[i,j] \wedge Y[i,j] = 1))$
  $i \leftarrow i + 1$
end for
$gc(R) \leftarrow \bigwedge_{j=1}^{m} (\bigvee_{i=1}^{n} (Y[1,j] = \ldots = Y[i-1,j] = Y[i+1,j] = \ldots = Y[n,j] = 0 \wedge Y[i,j] = 1))$
return $R \cup^d V_1$

Figure 7. The algorithm for subtracting mixed tables.

TABLE IX. THE RESULT OF **Student** $-^d$ *simplify*(**Student**)

name	*school*	*grade*	*local condition*
"John"	"UCLA"	y	$x = 1 \wedge \neg(\text{“}John\text{”} = n \wedge y = g \wedge ((x = 1 \wedge n = \text{“}John\text{”} \wedge g = y) \vee (x = 2 \wedge n = \text{“}Bob\text{”} \wedge g = \text{“}A\text{”})) \wedge Y[1,1] = 1)$
"Bob"	"UCLA"	"A"	$x = 2 \wedge \neg(\text{“}Bob\text{”} = n \wedge g = \text{“}A\text{”} \wedge ((x = 1 \wedge n = \text{“}John\text{”} \wedge g = y) \vee (x = 2 \wedge n = \text{“}Bob\text{”} \wedge g = \text{“}A\text{”})) \wedge Y[2,1] = 1)$

global condition: $(Y[1,1] = 1 \wedge Y[2,1] = 0) \vee (Y[1,1] = 0 \wedge Y[2,1] = 1)$

value of y	*prob.*
"A"	0.6
"B"	0.3
"C"	0.1

value of x	*prob.*
1	0.5
2	0.5

the local condition of only of the tuples can be satisfied. The formal definition of unifiable main parts is presented next.

Definition 4.4 (unifiable main parts): Two tuples have main parts that are unifiable if these main parts can become equivalent under some possible interpretation. We will write $unifiable(main(t_1), main(t_2))$ when this is the case.

Details are shown in Figure 8, where we use δ to denote the duplicate elimination operation. The following theorem is not hard to prove.

Algorithm 8 Evaluating $\delta(T)$

$V \leftarrow T$
for all $t_1, t_2 \in V$ **do**
  if *unifiable*(*main*(t_1),*main*(t_2)) and $simplify(lc(t_1) \equiv (t_2)) \not\equiv$ **false then**
    introduce a new variable x
    $lc(t_1) \leftarrow lc(t_1) \wedge ((main(t_1) = main(t_2)) \Rightarrow (x = 1))$
    $lc(t_2) \leftarrow lc(t_2) \wedge ((main(t_1) = main(t_2)) \Rightarrow (x = 2))$
  end if
end for
for all new variable x **do**
  Add a table for the distribution of x. The possible values are 1 and 2 with probability 0.5 each.
end for
return V

Figure 8. The algorithm for duplicate elimination.

We had to modify the definition of completeness because we allow the empty set (a.k.a. $\varnothing$) to be a possible representation of an m-table. We believe that this is intrinsic problem associated with monus under the closed world assumption. Table IX shows the result of applying the monus operation on the m-tables *simplify*(**Student**) and **Student**. Applying the *simplify* algorithm to the new table will produce the empty set.

F. Duplicate Elimination

The duplicate elimination algorithm simply checks for pairs of m-tuples that have main parts that are unifiable and local conditions that are not excluding. The algorithm then adds the restriction to the output table that states that if the two tuples indeed have the same main part under some interpretation, then

Theorem 4.6: The duplicate eliminating operation is closed, sound, and complete. The complexity of the operation is quadratic relative to the size of the table plus the time to execute the calls to the *simplify* function.

As an example, consider applying the duplicate eliminating operation on the table from Table VIII. The result is shown in Table X. We can apply the *simplify* function to the result to get back the **Student** table.

V. CONCLUSION AND FUTURE RESEARCH

We introduced the concept of a mixed table. This is a table that allows both random variables and linear constraints on them to be stored. To the best of our knowledge, we are the first paper to do so. We presented the semantics of a mixed table as

TABLE X. THE RESULT OF δ(**Student** $\cup^d$ **Student**)

name	*school*	*grade*	*local condition*
"John"	"UCLA"	y	$x = 1 \wedge z = 1$
"Bob"	"UCLA"	"A"	$x = 2 \wedge w = 1$
"John"	"UCLA"	y	$x = 1 \wedge z = 2$
"Bob"	"UCLA"	"A"	$x = 2 \wedge w = 2$

value of y	*prob.*
"A"	0.6
"B"	0.3
"C"	0.1

value of x	*prob.*
1	0.5
2	0.5

value of z	*prob.*
1	0.5
2	0.5

value of w	*prob.*
1	0.5
2	0.5

a set of relational tables, where a probability is associated with each representation. We extended the bag relational algebra from [8] to m-tables and showed that the new relational algebra is closed, sound, and complete. In summary, we believe that this paper can serve as a blueprint for a system for storing and querying m-tables.

As part of future research, we need to show how the different relational algebra operations can be performed efficiently. For example, indexes on the data can be used to execute the selection and join relational operations efficiently. We also need to follow the model from [8] and introduce algorithms for performing grouping and aggregation over m-tables.

REFERENCES

[1] L. Stanchev, "Bag Relational Algebra with Grouping and Aggregation over C-Tables with Linear Conditions," International Journal on Advances in Intelligent Systems, vol. 4, no. 3, 2010, pp. 258–272.

[2] D. Suciu, D. Olteanu, C. Re, and C. Koch, Probabilistic Databases. Morgan and Claypool Publishers, 2011.

[3] N. Dalvi, C. Re, and D. Sucu, "Probabilistic Databases: Diamonds in the Dirt," Communications of the ACM, vol. 52, no. 7, 2009, pp. 86–94.

[4] G. Grahne, The Problem of Incomplete Information in Relational Databases. Berlin: Springer-Verlag, 1991.

[5] R. Dechter, Constraint Networks. Encyclopedia of Artificial Intelligence, Second Edition, Wiley and Sons, 1992.

[6] J. Pearl, "Bayesian Networks: A Model of Self-Activated Memory for Evidential Reasoning," Seventh Annual Conference of the Cognitive Science Society, 1985, pp. 329–334.

[7] R. Mateescu and R. Dechter, "Mixed Deterministic and Probabilistic Networks," Annals of Mathematics and Artificial Intelligence, vol. 54, no. 1, 2008, pp. 3–51.

[8] L. Stanchev, "Querying Incomplete Information using Bag Relational Algebra," Proceedings of The Second International Conference on Information Process and Knowledge Management, 2010, pp. 110–119.

[9] J. Biskup, "A Formal Approach to null Values in Database Relations," Advances in Database Theory, 1981, pp. 299–341.

[10] E. Codd, "Understanding Relations (Installment 7)," Bulletin of ACM-SIGMOD, vol. 3, no. 4, 1975, pp. 23–28.

[11] ——, "Extending the Database Relational Model to Capture more Meaning," ACM Transactions on Database Systems, vol. 4, no. 4, 1979, pp. 397–434.

[12] J.Grant, "Null values in Relational Data Base," Information Processing Letters, vol. 6, no. 5, 1977, pp. 156–157.

[13] T. Imielinski and W. Lipski, "On Representing Incomplete Information in a Relational Data Base," Proceedings of the Seventh International Conference on Very Large Data Bases, 1981, pp. 388–397.

[14] ——, "Incomplete Information in Relational Algebra," Journal of Association of Computing, vol. 31, no. 4, 1984, pp. 761–791.

[15] L. Libkin and L. Wong, "Some Properties of Query Languages for Bags," Proceedings of Database Programming Languages, 1994, pp. 97–114.

[16] R. Reiter, "A Sound and Sometimes Complete Query Evaluation Algorithm for Relational Databases with Null Values," Journal of the ACM, vol. 33, no. 2, 1986, pp. 349–370.

[17] L. Yuan and D. Chiang, "A Sound and Complete Query Evaluation Algorithm for Relational Databases with Null Values," ACM Special Interest Group on Management of Data (SIGMOD), 1988, pp. 74–81.

[18] L. Libkin, "Query Languages Primitives for Programming with Incomplete Databases," Proceedings of the Fifth International Workshop on Database Programming Languages, 1995, pp. 1–13.

[19] P. Buneman, A. Jung, and A.Ohori, "Using Powerdomains to Generalize Relational Databases," Theoretical Computer Science, vol. 91, no. 1, 1991, pp. 23–55.

[20] E. Lefons, A. Silverstri, and F. Tangorra, "An Analytic Approach to Statistical Databases," International Conference on Very Large Data Bases, 1983, pp. 260–274.

[21] S. Ghosh, "Statistical Relational Tables for Statistical Database Management," IEEE Transactions on Software Engineering, vol. 12, no. 12, 1986, pp. 1106–1116.

[22] E. Gelenbe and G. Hebrail, "A Probability model of uncertainty in data bases," Second IEEE Conference on Data Engineering, 1986, pp. 328–333.

[23] R. Cavallo and M. Pittarelli, "The Theory of Probabilistic Databases," Proceedings of the Thirteen International Conference on Very Large Data Bases, 1987, pp. 71–81.

[24] D. Barbara, H. Garcia-Molina, and D. Porter, "The Management of Probabilistic Data," IEEE Transactions on Knowledge and Data Engineering, vol. 4, no. 5, 1992, pp. 487–502.

[25] T. Ge, A. Dekhtyar, and J. Gldsmith, "Uncertain Data: Representations, Query Processing, and Applications," in Advances in Probabilistic Databases. Springer-Verlag Berlin Heidelberg, 2013, pp. 67–108.

[26] R. Jampani, F. Xu, and M. Wu, "MCDB: A Monte Carlo Approach to Managing Uncertain Data," Proceeding of ACM Special Interest Group on Management of Data (SIGMOD), 2008, pp. 687–700.

[27] N. Fuhr and T. Rolleke, "A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems," ACM Transactions on Information Systems, vol. 15, no. 1, 1997, pp. 32–66.

[28] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widowm, "Trio: A System for Data, Uncertainty, and Lineage," Proceedings of Very Large Data Bases, 2006, pp. 1151–1154.

[29] N. N. Dalvi and D. Sucui, "Efficient Query Evaluation on Probabilistic Databases," Very Large Data Bases Journal, vol. 16, no. 4, 2007, pp. 523–544.

[30] P. Andritsos, A. Fuxman, and R. J. Miller, "Clean Answers over Dirty Databases: A Probabilistic Approach," Proceedings of the International Conference on Data Engineering, 2006, p. 30.

[31] L. Antova, C. Koch, and D. Olteanu, "MayBMS: Managing Incomplete information with Probabilistic World-set Decomposition," International Conference on Data Engineering, 2007, pp. 1479–1480.

[32] T. Imielinski and W. Lipski, "Incomplete Information in a Relational Data Base," Journal of ACM, vol. 31, no. 4, 1984, pp. 761–791.

[33] J. Lasssez and K. McAloon, "Applications of a Canonical Form of Generalized Linear Constraints," Journal of Symbolic Computations, vol. 13, 1992, pp. 1–24.

 ISBN: 978-1-61208-472-5