Analyzing Uncertain Tabular Data

Oliver Kennedy and Boris Glavic

Abstract

It is common practice to spend considerable time refining source data to address issues of data quality before beginning any data analysis. For example, an analyst might impute missing values or detect and fuse duplicate records representing the same real-world entity. However, there are many situations where there are multiple possible candidate resolutions for a data quality issue, but there is not sufficient evidence for determining which of the resolutions is the most appropriate. In this case, the only way forward is to make assumptions to restrict the space of solutions and/or to heuristically choose a resolution based on characteristics that are deemed predictive of "good" resolutions. Although it is important for the analyst to understand the impact of these assumptions and heuristic choices on her results, evaluating this impact can be highly non-trivial and time consuming. For several decades now, the fields of probabilistic, incomplete, and fuzzy databases have developed strategies for analyzing the impact of uncertainty on the outcome of analyses. This general family of uncertainty-aware databases aims to model ambiguity in the results of analyses expressed in standard languages like SQL, SparQL, R, or Spark. An uncertainty-aware database uses descriptions of potential errors and ambiguities in source data to derive a corresponding description of potential errors or ambiguities in the result of an analysis accessing this source data. Depending on technique, these descriptions of uncertainty may be either quantitative (bounds, probabilities), or qualitative (certain outcomes, unknown values, explanations of uncertainty). In this chapter, we explore the types of problems that techniques from uncertainty-aware databases address, survey solutions to these problems, and highlight their application to fixing data quality issues.

Oliver Kennedy
University at Buffalo, SUNY, e-mail: okennedy@buffalo.edu

Boris Glavic
Illinois Institute of Technology e-mail: bglavic@iit.edu

1 Introduction

Data quality is increasingly relevant to all facets of data management, from small-scale personal sensing applications to large corporate or scientific data analytics. In these and many other settings, sources of uncertainty include inaccuracies and failures of sensors, human data entry errors, systematic errors during fusion of data from heterogeneous sources, and many others. A prevalent problem when identifying data quality issues is that typically available information is insufficient to determine with certainty whether a fluke detected in the data is a quality issue or simply an unusual fact. For example, a high blood pressure value may be due to a medical problem or may as well be a measurement error — without further information it is impossible to distinguish between these two cases. Even if a quality issue can be detected with certainty this does not imply that we can find a unique or even any correct resolution for this issue. For instance, consider an employment dataset with records storing for each employee a unique identifier like a social security number (SSN), their name, and the department they work in. A single person might occur multiple times in this dataset if the person works for more than one department. If we assume that unique identifiers are truly unique, one type of data quality issue that is easy to detect (for this dataset) is when multiple records corresponding to the same identifier have different names[1]. While this condition is easy to check, it is not straightforward to repair a dataset with records that share identifiers but have conflicting names. For example, assume that our dataset contains two conflicting records `(777-777-7777, Peter Smith, Sales)` and `(777-777-7777, Bob Smith, Marketing)`. These records indicate that a person with SSN 777-777-7777 is working in both Sales and Marketing, and is either named Peter Smith or Bob Smith. The correct way to fix this conflict depends on what caused the error, and requires us to have additional information that is not readily available. If we assume that two people with the same SSN must have the same name, then there are four ways we could fix the error: 1. We could assume that one of the records was created mistakenly and delete it; 2. We could assume that our analysis will be minimally affected by this error and leave the records as-is; 3. We could assume that the name attribute of one (or both) of the records is incorrect and update them accordingly; and/or 4. We could assume that the SSN attribute of one (or both) of the records is incorrect, and update them accordingly. In absence of further information, there is no way for us to determine what the correct fix for the problem is.

[1] In database terminology, we would say that a functional dependency $id \rightarrow name$ holds for the dataset, i.e., the 'id' value of a record determines its name value. Or put differently, there are no two records that have the same SSN, but a different name. The intricacies of functional dependencies are beyond the scope of this paper. The interested reader is referred to database textbooks (e.g., [1]). Furthermore, see, e.g., [12] for how constraints like functional dependencies are used to repair data errors.

A typical approach for resolving this kind of ambiguity is to correlate the information in our dataset with externally available sources of high quality data, often called master data, to determine what fix to apply. For instance, we may have access to a reliable list of what name corresponds to which SSN. Such information is useful, because it allows us to make SSNs consistent with names. However, it is not sufficient for figuring out whether the issue arose from incorrect name(s) or from incorrect SSN(s) and, thus, whether the correct solution is to repair SSN or name value(s). This example demonstrates that even trivial data quality issues can have hard- or impossible-to-determine resolutions.

In a typical analytics workflow, problems like this are addressed at the start: a task we call *data curation*. Common approaches to data curation include: 1. If redundant data sources can be found, the curator can manually repair errors; 2. If records are irreperable, the curator may remove them from the dataset outright; 3. If removing records with errors would create bias or leave too few records, the curator can apply heuristic techniques (e.g., imputation) to repair errors. 4. If the errors are deemed to not affect the analysis, the curator may simply choose to not address them. For example, we might train a classifier to find records in our master data most similar to Peter and Bob Smith, and then use it to fix the name and/or SSN fields of our erroneous data set.

A common assumption is that once curation is complete, the data is "correct," at least to the point where specific analyses run over it will produce valid results. However, such assumptions can produce misleading analytical results. The most straightforward of these are the consequences of incorrect choices during curation — for example selecting a classifier that is inappropriate for the available data. However, there is a more subtle possibility: The data being curated might be insufficient or of insufficient quality — regardless of curation technique — to support some or all of the analyst's goals.

Existing data management systems: Relational Databases, Graph Databases, Spark, NoSQL Systems, R, and others, are built on the assumption that data is exact. These systems can not distinguish low-confidence information derived of incomplete data or heuristic guesswork from high-confidence information grounded in verifiable data fused from multiple redundant data sources. In extreme cases, such misinterpretations ruin lives. For example, credit report errors can severely limit a person's access to financial services and job opportunities.

Example 1. Consider an example bank's loan database shown in Figure 1. Table `customers` stores the SSN, name, income, and assets for each customer. Furthermore, we record for each customer the number of mortgages this customer has and whether they own property or not. Table `applications` stores loan applications (a customer's SSN and the amount of loan they requested). The bank uses the following formula to determine the maximum amount $Max_{\$}$ they are willing to loan to a customer.

$$Max_{\$} = 0.3 \cdot income + 0.2 * assets - 10,000 \cdot numMortgages + 40,000 \cdot ownsProperty$$

The bank might use a SQL query such as the one shown below to determine which loan applications it should grant and which applications will be rejected.

```
SELECT SSN, name, CASE WHEN maxAllow >= requestedAmount
                       THEN 'yes' ELSE 'no'
                  END AS grantLoan
FROM (SELECT 0.3 * income + 0.1 * assets - 10000 * numMortgages
        + CASE WHEN ownsProperty THEN 20000 ELSE 0 END AS maxAllow,
        loanAmount AS requestedAmount
      FROM customer c, applications a WHERE c.SSN = a.SSN) sub
```

For readers unfamiliar with SQL, this query (1) retrieves the customer information for each loan request, then (2) computes $Max_{\$}$ based on the customer information, and (3) returns the SSN and the name of the customer requesting the loan as well as a column `grantLoan` whose value is `yes` if the amount requested is lower than $Max_{\$}$ or *no* otherwise.

Unfortunately, the customers table contains missing values (in SQL the special value **NULL** is used to denote that the value of an attribute is missing). For example, we do not know whether Alice owns property or not, or what Peter's income is. These missing values prevent us from determining whether Alice's loan should be approved or not. As mentioned above, one way to resolve this issue is to train a classifier, such as a decision tree. The classifier is trained on records that have all their attribute values. Given the values of the other attributes (`SSN`, `name`, `income`, `assets`, and `#mortgages`) for a record, the trained classifier predicts the missing value of the `ownsProperty` attribute.

Classifier Result for Alice

ownProperty	probability
yes	0.4
no	0.6

Query Result

SSN	name	grantLoan
777-777-7777	Alice Alison	no
111-111-1111	Peter Peterson	yes

Assume that the classifier predicts that the value of `ownsProperty` for Alice's record is yes with 40% probability and no with 60% probability. The **NULL** value is replaced with *no*, the value with the highest probability. If the dataset curated in this way is used to determine Alice's elegibility to receive the loan, then the bank would deny her loan application — even if she does in fact own property and should be eligible. In this case, the bank does not have enough information to decide whether to give Alice a loan, but by treating imputed missing values as fact, this lack of information is obscured.

As we will explain in more detail in the remainder of this chapter, uncertainty-aware approaches to data management can expose this lack of information. In the example, this would allow the bank's computer to produce a third response: "I don't know", exposing the uncertainty in the analytical result. Depending on what technique for uncertain data management

customers

SSN	name	income	assets	numMortgages	ownsProperty
777-777-7777	Alice Alison	$60,000	$200,000	0	**NULL**
333-333-3333	Bob Bobsen	$102,000	$15,000	0	no
111-111-1111	Peter Petersen	**NULL**	$90,000	1	yes
555-555-5555	Arno Arnoldson	$95,000	$30,000	0	yes

applications

SSN	loanAmount
777-777-7777	$90,000
111-111-1111	$10,000

Fig. 1: Example for resolving a data quality issue and how this affects the trustworthiness of analysis results. The input data contains missing values and needs to be imputed.

is applied, it might also be feasible to determine exactly what information is necessary to produce a definitive answer to the bank's question.

Uncertain data management techniques can help to expose implicit biases introduced during data curation, as well as the impact that these biases have on the quality and trustworthiness of analytical results. In this chapter, we provide an introduction to uncertain data management techniques developed by the database community and their use in data curation and analysis. While there are excellent surveys on probabilistic data management, e.g., [39] and [8], these surveys aim to describe the technical challenges involved. In contrast, we give a more goal-oriented introduction to the topic targetted at helping practitioners identify suitable techniques for their needs, and to clarify the connection to practical applications of these techniques for data quality.

Making uncertainty a first-class primitive in data management has been the goal of so-called *uncertain* and *probabilistic query processing* [39], or *PQP*. There have been significant efforts in this space, aiming to produce efficient algorithms for some of the most computationally challenging tasks involved in managing uncertain data [7]. While great progress has been made, some tasks in PQP are too computationally intensive to be widely applied. Thus, having practical applications in mind we also survey light-weight approaches which produce less detailed and/or approximate descriptions of uncertainty, at the benefit of reduced computational complexity. Though they are approximations, these simpler representation of uncertainty are also often easier to interpret than more complex models.

The remainder of this chapter is organized as follows. We introduce core concepts of uncertain data management in Section 2. We then cover data models that encode uncertainty in one form or another in Section 3. Section 4 covers methods for computing the results of queries and evaluating general

programs over uncertain data. Afterwards, we discuss methods for presenting uncertainty to endusers in Section 5. Finally, we conclude in Section 6.

2 Core Concepts: Incomplete and Probabilistic Databases

We begin with the broadest assumptions about the workflows applied by an analyst, before narrowing our scope to particular models of computation (mostly declarative query languages) when discussing specific solutions. Specifically, we assume that the analyst has specified her analysis as a program Q expressed in some suitable form such as an imperative programming language (e.g., Python or C) or a database query language (e.g., SQL or Spark). When applied to some dataset D, this program produces a result $Q(D)$. For instance, for our running example, the program Q is the SQL query shown in Figure 1, D is the cleaned version of the loan dataset (after imputing the missing `ownsProperty` value of Alice's record), and $Q(D)$ is the query result shown in Figure 1.

2.1 Possible World Semantics

A well-established model for uncertainty in such an analysis is the so-called possible world semantics. Under the possible worlds semantics we forsake the existence of single deterministic dataset and instead consider a set of *possible* datasets, the possible worlds. Each possible world is a dataset that could exist under certain assumptions. Formally, an uncertain dataset is a (potentially infinite) set of datasets $\mathcal{D}$. We refer to $\mathcal{D}$ as an *incomplete database*, and its members as *possible worlds*.

Example 2. Continuing with our running example, recall that we used a classifier to impute the missing `ownsProperty` attribute of Alice's record. In the previous example we did pick the most likely value predicted by the classifier (no) and saw how this incorrect choice caused Alice's loan to be rejected. The heuristic underlying this choice is the classifier; We assume that it will always pick the correct replacement value — in general a quite strong assumption. Instead we can model the output of the missing value imputation as a set of possible worlds $\mathcal{D} = \{D_1, D_2\}$. In this example there are two worlds that are shown in Figure 2: either Alice does not own property (possible world D_1 shown on top) or Alice does own property (possible world D_2 shown on the bottom). In this possible world model, it is evident that we are uncertain about whether Alice owns property or not, as the only difference between the two possible worlds is the ownsProperty attribute value for Alice's record.

Possible World D_1 ($p = 0.6$)
customers

SSN	name	income	assets	#mortgages	ownsProperty
777-777-7777	Alice Alison	$60,000	$200,000	0	no
333-333-3333	Bob Bobsen	$102,000	$15,000	0	no
111-111-1111	Peter Petersen	**NULL**	$90,000	1	yes
555-555-5555	Arno Arnoldson	$95,000	$30,000	0	yes

Possible World D_2 ($p = 0.4$)
customers

SSN	name	income	assets	#mortgages	ownsProperty
777-777-7777	Alice Alison	$60,000	$200,000	0	yes
333-333-3333	Bob Bobsen	$102,000	$15,000	0	no
111-111-1111	Peter Petersen	**NULL**	$90,000	1	yes
555-555-5555	Arno Arnoldson	$95,000	$30,000	0	yes

Fig. 2: Possible Worlds Representation of the Cleaned Loan Dataset Created Based on the Possible Imputed Values for the Missing ownsProperty Value Predicted by the Classifier.

When an analysis program Q is evaluated over an incomplete database $\mathcal{D}$, the result is not one, but a set of possible results: the set of all results that could be obtained by evaluating Q in some possible world:

$$Q(\mathcal{D}) := \{\ Q(D)\ \mid D \in \mathcal{D}\ \}$$

That is, the result of evaluating an analysis program Q over an incomplete database is itself an incomplete database.

Example 3. Evaluating our loan query over the incomplete database from Figure 2, we get two possible results shown below. We are sure that Peter should be granted a loan, but there are two possible outcomes for Alice's loan application. If Alice owns property (possible world D_2) then we would consider her to be eligible for this loan (possible result $Q(D_2)$). Otherwise, (possible world D_1) we should reject her loan application $Q(D_1)$.

$Q(D_1)$

SSN	name	grantLoan
777-777-7777	Alice Alison	no
111-111-1111	Peter Peterson	yes

$Q(D_2)$

SSN	name	grantLoan
777-777-7777	Alice Alison	yes
111-111-1111	Peter Peterson	yes

2.2 Certain and Possible Records

It is often useful to reason about a dataset D as a collection of records $r \in D$, or a table. When using this model, we also treat analytical results as sets of

records $r \in Q(D)$. We then may want to reason about what facts (records) we know for certain to be true, what facts are potentially true, and which facts are known to be untrue. For instance, in our running example we know with certainty that Peter's loan should be granted, since regardless of which possible world represents the true state of the world, the loan application will be granted. For incomplete databases, we say that a record is *certain* if the record appears in every possible world. Note that, because the result of analysis over an incomplete database is again an incomplete database, we can apply the same concept to analysis results. A record is certainly in the result of an analysis if it is present in the result irrespective of which possible world correctly describes the true state of the world. Formally, a record r is *certain* in an incomplete database $\mathcal{D}$ if $\forall D \in \mathcal{D} : r \in D$. We use $\textbf{certain}(\mathcal{D})$ to denote the set of all certain records in $\mathcal{D}$. Analogously, the certain answers to a question $Q(\mathcal{D})$, which we write as $\textbf{certain}(Q(\mathcal{D}))$ are the set of all certain records in the incomplete result $Q(\mathcal{D})$.

$$r \textbf{ is certainly in } \mathcal{D} \;:=\; \forall D \in \mathcal{D} : r \in D$$

$$\textbf{certain}(Q(\mathcal{D})) = \{\; r \mid \forall D \in \mathcal{D} : r \in D \;\}$$

Conversely, we say that a record is *possible* if the record appears in at least one possible world (resp., possible result). The possible answers of a question ($\textbf{possible}(Q(\mathcal{D}))$) are defined symmetrically.

$$r \textbf{ is possibly in } \mathcal{D} \;:=\; \exists D \in \mathcal{D} : r \in D$$

$$\textbf{possible}(Q(\mathcal{D})) = \{\; r \mid \exists D \in \mathcal{D} : r \in D \;\}$$

Example 4. For instance, `(111-111-1111,Peter Peterson,yes)` is a certain answer in our running example since it is in the result for every possible world, but `(777-777-7777,Alice Alison,yes)` is not since this record is not in $Q(D_1)$. Conversely, `(777-777-7777,Alice Alison,yes)` is possible while `(777-777-7777,Alice Peterson,yes)` is not.

2.3 Multisets

Many database systems use *bag*s (multisets) of records to represent data. That is, bag databases allow for multiple records that have all attribute values in common. For example, if we allow multiple loan applications by the same person, then it may be possible that two applications from the same person are requesting the same amount, i.e., are duplicates.

Example 5. Consider the bag table `applications` shown below. Here both Alice (SSN 777-777-7777) and Peter (SSN 111-111-1111) have submitted two

applications. Alice's applications are both for a loan of $90,000, i.e., there is a duplicate of the record `(777-777-7777, $90,000)` in the applications table.

SSN	loanAmount
777-777-7777	$90,000
777-777-7777	$90,000
111-111-1111	$10,000
111-111-1111	$3,000

We write $D[r]$ (resp., $Q(D)[r]$, $R[r]$) to denote the number of occurrences, or the multiplicity of record r in database D (resp. result $Q(D)$ or record set R). We can now define the multiplicities of an uncertain record as a set of multiplicities for each possible world.

$$\mathcal{D}[r] = \{\ D[r]\ \mid D \in \mathcal{D}\ \}$$

Libkin et. al [19] generalize the notion of certain and possible results to reason about bounds on these multisets (or bags).

$$\textbf{certain}(\mathcal{D}[r]) = \texttt{min}(\mathcal{D}[r]) \qquad \textbf{possible}(\mathcal{D}[r]) = \texttt{max}(\mathcal{D}[r])$$

Observe that when a multiset $\mathcal{D}$ encodes a set (i.e., when $\mathcal{D}[r] \in \{0,1\}$), multiset possible and certain answers behave as their set-based counterparts.

2.4 Incorporating Probability

An incomplete database may be supplemented with a probability measure $P : \mathcal{D} \to [0,1]$ over the set of possible worlds requiring that $\sum_{D \in \mathcal{D}} P(D) = 1$. The pair $\langle\ \mathcal{D}, P\ \rangle$ of incomplete database and probability measure is called a probabilistic database [16]. When a question Q is asked of an probabilistic database $\langle\ \mathcal{D}, P\ \rangle$, we can derive a marginal distribution over the set of possible results $\{\ R_i\ \}$:

$$p[Q(\mathcal{D}) = R_i] := \sum_{D \in \mathcal{D}\ s.t.\ Q(D) = R_i} P(D)$$

When the result is a set (resp., bag) of records, we derive the marginal probability of any possible record in the result similarly (where $M \in \mathbb{N}$ is a record multiplicity).

$$p[r \in Q(\mathcal{D})] := \sum_{D \in \mathcal{D}\ s.t.\ r \in Q(D)} P(D)$$

$$p\big[Q(\mathcal{D})[r] = M\big] := \sum_{D \in \mathcal{D}\ s.t.\ Q(D)[r] = M} P(D)$$

We will sometimes use $p(r)$ as a notational shortcut for $p[r \in Q(\mathcal{D})]$ when Q and $\mathcal{D}$ are understood from the context.

Example 6. Recall that the classifier we have trained predicts that Alice owns property with 40% probability and that she does not own property with 60% probability. Thus, possible world D_1 has probability $p(D_1) = 0.6$ and possible world D_2 has probability $p(D_2) = 0.4$. Then based on this probability distribution of the input possible worlds, the probabilities of the two possible query results are determined. The probability of the result where Ann's loan application is rejected is 0.6 while the one where Ann's loan is granted is 0.4. Given this result, we can compute the marginal probability of an analysis result record, i.e., the likelihood that this particular record exists in the result as the sum of the probabilities of possible worlds containing this record. Let r_1 be the record corresponding to the loan granted to Peter. Since this record appears in both possible worlds we get $p(r_1) = p(D_1)+p(D_2) = 0.4+0.6 = 1$. That is, the record has probability 1.

Observe that there is a strong connection between the concepts of certain/-possible records and the marginal probability of a record. Given a record r, we have:

$$p[r \in Q(\mathcal{D})] = 1 \Leftrightarrow r \in \textbf{certain}(Q(\mathcal{D}))$$
$$p[r \in Q(\mathcal{D})] > 0 \Leftrightarrow r \in \textbf{possible}(Q(\mathcal{D}))$$

Records which are certain (occur in every possible world) must have marginal probability 1 (the probabilities of all possible worlds have to sum up to 1) while records that are possible must have a non-zero probability (they occur in at least one possible world).

3 Uncertainty Encodings

Possible worlds provide us with a convenient, intuitive model for uncertain data processing that is independent on the choice of language for expressing analysis tasks. However, the full set of possible worlds (and the corresponding probability distributions) may be extremely large (or even infinite). This may not be obvious from our running example, but observe that in our example there was only one missing value to impute and we only had two possible replacements for this missing value. Consider a moderately sized version of our running example dataset with 100k records and assume that there 100 missing `ownsProperty` values. Then there are 2^{100} possible worlds - one for each choice of values for these 100 missing values. To make analysis of uncertain data feasible or even to store such a large number of possible worlds, a more compact representation is needed. A variety of uncertain data encodings have

been developed that compactly represent sets of possible worlds. We now focus our discussion exclusively on representations of collections (sets or bags) of records. Broadly, we categorize encodings of uncertain collection datasets into two classes: (1) **Lossless** encodings, which exploit independence properties to factorize the set of possible worlds, and (2) **Sampled** encodings, which represent a finite subset of the possible worlds at the cost of information loss.

3.1 Lossless, Factorized Encodings for Incomplete Data

Independence can be exploited to create compact, factorized representations of uncertain data. For example, consider a record r_1 that is present in exactly half of the possible worlds of $\mathcal{D}$, where the remaining possible worlds (modulo r_1) are identical. Writing $\mathcal{D}'$ to denote the possible worlds without r_1 ($\mathcal{D}' = \{\ D \mid D \in \mathcal{D}, r_1 \notin D\ \}$), we can call record r_1 independent iff

$$\mathcal{D} = \mathcal{D}' \cup \{\ \{r_1\} \cup D' \mid D' \in \mathcal{D}'\ \}$$

Intuitively, the above condition checks that the choice of including r_1 or not has no effect on the inclusion of any the other possible records. A natural consequence of r_1's independence is that the possible worlds encoding of $\mathcal{D}$ need to store two full copies of $\mathcal{D}'$. Lossless encodings exploit such redundancy caused by independence to create more compact representations where this redundancy is factorized out. We will outline three lossless encodings: Tuple-Independent Databases, Disjoint-Independent Databases, and C-Tables, each a more expressive generalization of the previous one. For instance, in our running example the record encoding that Peter's loan is granted appears in both possible worlds while Alice's record, even through it also appears in both possible worlds, occurs with different `ownsProperty` values. Based on this observation a more compact representation of D_1 and D_2 from our running example is as a single database containing both Peter's and Alice's record, but to record separately that there are two options (yes and no) for the `ownsProperty` attribute value for Alice's record.

3.1.1 Tuple-Independent Databases

We first consider an encoding that is only applicable if the following simple, but strong condition holds: all records are independent of one another. An incomplete database that satisfies this constraint is called a tuple-independent incomplete database [4, 40]. Note that *tuple* is the formal term used to denote records in a database. A tuple-independent incomplete database can be represented as a collection of records $\mathbb{D}_{TI} \in dom(r) \times \mathbb{B}$, where each record

is annotated with a boolean attribute that indicates whether (or not) it is certain. Note that here $dom(r)$ denotes the domain of records, i.e., the set of all possible records. That is:

$$\textbf{certain}(\mathbb{D}_{TI}) = \{\ r\ \mid \langle\ r, \texttt{TRUE}\ \rangle \in \mathbb{D}_{TI}\ \}$$

$$\textbf{possible}(\mathbb{D}_{TI}) = \{\ r\ \mid \langle\ r, \_\ \rangle \in \mathbb{D}_{TI}\ \}$$

The incomplete database $\mathcal{D}$ represented by $\mathbb{D}_{TI}$ is defined as the set of all databases that are subsets of the set of all possible records and supersets of the set of certain records.

$$\mathcal{D} = \{\ D\ \mid D \subseteq \textbf{possible}(\mathbb{D}_{TI}) \wedge D \supseteq \textbf{certain}(\mathbb{D}_{TI})\ \}$$

The second requirement ($D \supseteq \textbf{certain}(\mathbb{D}_{TI})$) is based on the fact taht certain records appear in every possible world. Thus, any possible world represented by $\mathbb{D}_{TI}$ is a superset of the set of certain records as well.

Note that this type of factorization can not be used to compress the incomplete database from our running example, because the two versions of Alice's record are not independent of each other (a possible world including the tuple recording that Alice owns property cannot also contain the tuple recording that Alice does not own property).

Example 7. To illustrate tuple-independent databases consider a modified version of our running example where Alice's record only exists if she owns property. This modified version of our running example can be represented as a tuple-independent incomplete database (Peter's record is certain while the tuple storing that Alice does own property is only possible) as shown below:

SSN	name	grantLoan	isCertain
777-777-7777	Alice Alison	yes	_
111-111-1111	Peter Peterson	yes	TRUE

When using probabilities, we make a similar assumption that the probabilities of individual tuples are independent. Accordingly, for any probabilistic database $\langle\ \mathcal{D}, P\ \rangle$ fulfilling this condition, we can define a tuple-independent encoding $\mathbb{D}_{TIP} \in dom(r) \times [0, 1]$ by annotating each tuple with its probability instead of a boolean value:

$$\mathbb{D}_{TIP} = \{\ \langle\ r, P(r \in \mathcal{D})\ \rangle\ \mid r \in \textbf{possible}(\mathbb{D}_{TI})\ \}$$

In the probabilistic representation, certain answers are those records with a probability of 1, and possible answers are any records with a nonzero probability.

$$\textbf{certain}(\mathbb{D}_{TIP}) = \{\ r\ \mid \langle\ r, 1.0\ \rangle \in \mathbb{D}_{TIP}\ \}$$

$$\textbf{possible}(\mathbb{D}_{TIP}) = \{\ r\ \mid \langle\ r, p\ \rangle \in \mathbb{D}_{TIP} \wedge (p > 0)\ \}$$

Observe that a tuple-independent incomplete or probabilistic database with n tuples represents 2^n possible worlds: for each tuple we can either choose the tuple to be included in the possible world or not and all these choices are independent of each other. There are n boolean decisions resulting in 2^n possible options. That is, a tuple-independent incomplete or probabilistic database can be exponentially more concise than its possible world representation.

3.1.2 Disjoint-Independent Databases

The tuple-independent model assumes that records are entirely independent, but says nothing about the contents of those records. It is assumed that each record is identical in all possible worlds where it appears. The disjoint-independent model of incomplete databases (sometimes called x-tuples) generalizes tuple-independent databases by allowing a record to take multiple forms in different possible worlds. Under this model, an *x-tuple* $\mathbf{r}$ is simply a set of records $\{ r_1, \ldots, r_N \}$ called its *instantiations*. We say that $\mathbf{r}$ is disjoint-independent in database $\mathcal{D}$ iff we can define a subset of the possible worlds $\mathcal{D}' = \{ D \mid (D \in \mathcal{D}) \wedge (\mathbf{r} \cap D = \emptyset) \}$ that do not contain elements of $\mathbf{r}$, such that $\mathcal{D}$ can be defined as the cartesian product of $\mathbf{r}$ and $\mathcal{D}'$. That is, $\mathbf{r}$ is disjoint independent iff:

$$\mathcal{D} = \{ D \cup \{r\} \mid D \in \mathcal{D}' \wedge r \in \mathbf{r} \} \quad \textbf{or} \quad \mathcal{D} = \{ D \cup \{r\} \mid D \in \mathcal{D}' \wedge r \in \mathbf{r} \} \cup \mathcal{D}'$$

Note the two definitions: In the former case, *some* instantiation of $\mathbf{r}$ appears in all possible worlds, while in the latter some set of possible worlds $\mathcal{D}'$ do not contain any instantiation of $\mathbf{r}$. Accordingly, in the former case $\mathbf{r}$ is certain, while in the latter it is merely possible. Note that if an x-tuple $\mathbf{r}$ is certain in a disjoint-independent database, this x-tuple may have different values in different possible worlds. Applying our previous definition of certainty which requires that a record r appears in all possible worlds to disjoint-independent databases a record r is certain if there exists an x-tuple $\mathbf{r} = \{r\}$.

Observe that disjoint-independent databases generalize tuple-independent databases in the following sense: a tuple-independent database can be modelled as a disjoint-independent database where each x-tuple has a single instantiation $\mathbf{r} = \{r\}$ for certain records and $\mathbf{r} = \{r, \bot\}$ for records that are not certain. However, the opposite direction does not hold, there are disjoint-independent databases that cannot be represented as tuple-independent incomplete databases, e.g., a database with two possible worlds $D_1 = \{r_1\}$ and $D_2 = \{r_2\}$ where $r_1 \neq r_2$ can be encoded using a single x-tuple $\mathbf{r} = \{r_1, r_2\}$. However, this incomplete database cannot be encoded as a tuple-independent database.

As before, we define an encoding $\mathbb{D}_{DI} \in 2^{2^{dom(r) \cup \{\bot\}}}$ for any database known to contain exclusively disjoint-independent records. Specifically, the encoding is a collection of x-tuples (sets of records). The presence of a special

distinguished value $\perp$ in an x-tuple indicates that the x-tuple is not certain. Hence, the incomplete database $\mathcal{D}$ corresponding to $\mathbb{D}_{DI}$ is defined as follows (using $\prod$ to denote a cartesian product of sets).

$$\mathcal{D} = \prod_{\mathbf{r} \in \mathbb{D}_{DI}} \{\ \hat{r} \mid r \in \mathbf{r}\ \} \qquad \textbf{where} \qquad \hat{r} = \begin{cases} \{\} & \text{if } r = \perp \\ \{r\} & \text{otherwise} \end{cases}$$

Example 8. Our original running example database can be encoded as a disjoint-independent database as shown below consisting of two x-tuples $\mathbf{r_1}$ and $\mathbf{r_2}$ where $\mathbf{r_1}$ (Peter's record) is certain and has one instantiation r_{1_1} while $\mathbf{r_2}$ has two instantiations (Alice owns property or not) r_{2_1} and r_{2_2}:

record	**instantiation**	**SSN**	**name**	**grantLoan**
r_1	r_{1_1}	111-111-1111	Peter Peterson	yes
r_2	r_{2_1}	777-777-7777	Alice Alison	yes
	r_{2_2}	777-777-7777	Alice Alison	no

The disjoint-independent encoding of incomplete databases can be extended to also support probabilistic databases. A disjoint-independent probabilistic database $\mathbb{D}_{DIP} : 2^{2^{dom(r)}} \rightarrow (2^{dom(r)} \rightarrow [0,1])$ is a function that map every possible x-tuple $\mathbf{r}$ (a subset of $dom(r)$) to a probability mass function over $\mathbf{r}$ (a function associating a probability from $[0,1]$ with individual instantiations of x-tuple $\mathbf{r}$). Note that this does not mean that we require that every possible x-tuple $\mathbf{r}$ exists. For non-existing x-tuples we would set the probability of all of its instantiations r_i to 0. While the domain $dom(r)$ of the function $\mathbb{D}_{DIP}$ that encodes a disjoint-independent probabilistic database may be infinite, $\mathbb{D}_{DIP}$ is finitely representable as long as each possible world is finite, which is a typically assumed to be the case. A finite representation is achieved by only recording the output of function $\mathbb{D}_{DIP}$ for input records with a total probability mass larger than 0. We use $p_{\mathbf{r}}$ to denote the probability distribution associated by $\mathbb{D}_{DIP}$ to an x-tuple $\mathbf{r}$, i.e., $p_{\mathbf{r}} = \mathbb{D}_{DIP}(\mathbf{r})$. Observe that, we have eliminated the need for a distinguished element $\perp$ to denote the x-tuple's absence by allowing its probability mass function to sum to less than 1. That is, we define:

$$P[\mathbf{r} \in \mathcal{D}] = \sum_{r \in \mathbf{r}} p_{\mathbf{r}}(r) \qquad \textbf{and} \qquad P[\mathbf{r} = r] = p_{\mathbf{r}}(r)$$

These probability mass functions give us natural definitions for both certain and possible records:

$$\textbf{certain}(\mathbb{D}_{DIP}) = \left\{ \mathbf{r} \;\middle|\; \sum_{r \in \mathbf{r}} p_{\mathbf{r}}(r) = 1 \right\}$$

$$\textbf{possible}(\mathbb{D}_{DIP}) = \left\{ \mathbf{r} \;\middle|\; \sum_{r \in \mathbf{r}} p_{\mathbf{r}}(r) > 0 \right\}$$

Again, for a record r to be certain in the sense we defined for possible world semantics, the singleton $\mathbf{r} = \{r\}$ has to be mapped to the probability distribution $p_{\mathbf{r}}(r) = 1$ by $\mathbb{D}_{DIP}$.[2] While the representation of an incomplete database as a tuple-independent database is unique this does not hold for disjoint-independent database since the same instantiation may occur in different tuples.

3.1.3 C-Tables

Like the tuple-independent model, the disjoint-independent model can not capture arbitrary correlations between records. Originally proposed by Imielinski and Lipski, the C-Tables model [21] allows incomplete databases to be factorized, without requiring that their records be independent or otherwise uncorrelated. We use $\mathbb{V}$ to denote an alphabet of variable symbols $v \in \mathbb{V}$. A C-Tables $\mathbb{D}_C$ is a collection of records $\langle\, r, f_r(v_1, \ldots, v_N) \,\rangle \in \mathbb{D}_C$, where every record r is annotated with a boolean expression f_r over variables $v_1 \ldots v_N \in \mathbb{V}$. This expression is sometimes termed a *local condition*. The set of possible worlds defined by a C-table is based on assignments $\alpha : \mathbb{V} \to \mathbb{B}$ of boolean values to each variable. We use $\mathcal{A}$ to denote the set of all such assignments. Specifically, for each assignment $\alpha \in \mathcal{A}$ there exists a possible world D_α defined by this assignment as the set of records in $\mathbb{D}_C$ that are annotated with an expression that evaluates to true after replacing variables with their values assigned by α.

$$D_\alpha = \{\; r \mid \langle\, r, f_r(v_1, \ldots, v_N) \,\rangle \in \mathbb{D}_C \wedge f(\alpha(v_1), \ldots \alpha(v_N)) \;\}$$

Given some boolean expression $F(v_1, \ldots, v_K)$, termed the *global condition*[3], the full set of possible worlds is defined by the set of assignments that cause F to evaluate to true:

$$\mathcal{D} = \{\; D_\alpha \mid \alpha \in \mathcal{A} \wedge F(\alpha(v_1), \ldots, \alpha(v_K)) \;\}$$

[2] The reader may wonder whether it is possible to encode a certain record r as multiple x-tuples that all have r as an instantiation and where for each such x-tuple $\mathbf{r}$ we have $p_{\mathbf{r}}(r) < 1$. However, recall that x-tuples are assumed to be independent of each other. Thus, there would exist a possible world with a non-zero probability that does not contain r constructed by choosing an instantiation $r' \neq r$ or no instantiation for every x-tuple $\mathbf{r}$ with $r \in \mathbf{r}$.

[3] Note that global conditions are not strictly necessary for expressive power, but they may allow for a more compact/convenient representation of a probabilistic database.

C-tables are more expressive than the tuple-independent and disjoint-independent databases. In fact, any finite set of possible worlds can be encoded as a C-table.[4]

Probabilistic C-Tables. This representation admits a straightforward extension to the probabilistic case, originally proposed by Green et. al. [16]. This approach defines a probability distribution $P : \mathcal{A} \to [0, 1]$ over the space of assignments.[5] A probabilistic C-Table (or PC-Table) is defined as a pair of database and probability distribution $\mathbb{D}_{PC} = \langle\, \mathbb{D}_C, P \,\rangle$ Hence, the probability of a database and record can be defined as typical for possible worlds semantics:

$$P[D_\alpha \in \langle\, \mathbb{D}_C, P \,\rangle] = P(\nu) \qquad P[r \in \langle\, \mathbb{D}_C, P \,\rangle] = \sum_{\alpha \in \mathcal{A} : r \in D_\alpha} P(\alpha)$$

The distribution P can be encoded using any standard approach for compactly encoding multivariate distributions, such as a graphical model [34].

Example 9. Continuing the running example, we can model the analysis result as a C-Table. There is one uncertain decision that affects the set of possible worlds: Whether or not Alice owns property. We define a single boolean variable v_1 to denote the outcome of this decision. Records in the C-Table encoding the result are annotated with boolean expressions ϕ over $\mathbb{V} = \{\, v_1 \,\}$:

Query Result (Simple C-Table)

SSN	name	grantLoan	ϕ
777-777-7777	Alice Alison	no	v_1
777-777-7777	Alice Alison	yes	$\neg v_1$
111-111-1111	Peter Peterson	yes	**T**

The two possible assignments $\{\, v_1 \mapsto \mathbf{T} \,\}$ and $\{\, v_1 \mapsto \mathbf{F} \,\}$ define the two possible worlds. A separately provided (joint) distribution over the variable(s) in $\mathbb{V}$ assigns a probability to each possible world.

$$p(\alpha) = \begin{cases} 0.6 & \textbf{if } \alpha = \{\, v_1 \mapsto \mathbf{T} \,\} \\ 0.4 & \textbf{if } \alpha = \{\, v_1 \mapsto \mathbf{F} \,\} \end{cases}$$

[4] Consider an incomplete database $\mathcal{D}$ with 2^n possible worlds $D_1 \ldots D_{2^n}$. (the construction has to be modified slightly is the number of possible worlds is not a power of 2). Then we use n variables: $v_1, \ldots, v_n$. An assignment to these variables is interpreted as a number i in binary identifying one possible world D_i. For example, if there are $4 = 2^2$ possible worlds, then we would use two variables v_1 and v_2 and the assignment $v_1 \mapsto \mathbf{T}$ and $v_2 \mapsto \mathbf{F}$ represents the possible world $1 \cdot 2^1 + 0 \cdot 2^0 = 2$. The database constructed contains all records that are possible in $\mathcal{D}$. For an assignment α, let $n(\alpha)$ denote the number encoded by α. Then the local condition for record r is $\bigvee_{\alpha : r \in D_{n(\alpha)}} \bigwedge_{j : \alpha(v_j) = \mathbf{T}} v_j$.

[5] Note that [16] used per variable distributions which is less general.

Each possible world contains only records with boolean expressions that are true under the corresponding assignment. Hence the first two rows (with conditions v_1 and $\neg v_1$ respectively) are mutually exclusive.

Non-Boolean Variables and Assignments. For C-Tables to efficiently model a disjoint-independent database, it is necessary to create local conditions that alternate between mutually exclusive options. Such conditions can be modeled with boolean formulas, as in Example 9.[6] However, it is often both convenient and more efficient to express alternatives with a single integer- or real-valued variable. In this form, records are still annotated with boolean expressions, albeit over comparisons ($=, <, \leq$, etc...) over variables.

Example 10. Continuing the example, we could express the same result using integer-valued variables. The result C-Table and corresponding distribution are as follows:

Query Result (Integer-Valuation C-Table)

SSN	name	grantLoan	ϕ
777-777-7777	Alice Alison	no	$(v_1 = 1)$
777-777-7777	Alice Alison	yes	$(v_1 = 2)$
111-111-1111	Peter Peterson	yes	**T**

$$p(\alpha) = \begin{cases} 0.6 & \textbf{if } \alpha = \{\ v_1 \mapsto 1\ \} \\ 0.4 & \textbf{if } \alpha = \{\ v_1 \mapsto 2\ \} \end{cases}$$

Taking the process even further, we can replace attributes with placeholders (often called "labeled" nulls or skolem terms) indicating that their value is to be given by the valuation. The result is an even more compact representation, as records that were previously conditionally in the result can now be treated as certain.

Example 11. Having two assignments with nonzero probability: $\{\ v_1 \mapsto$ `'no'` $\}$ and $\{\ v_1 \mapsto$ `'yes'` $\}$, we can replace the result C-table as follows:

Query Result (General C-Table)

SSN	name	grantLoan	ϕ
777-777-7777	Alice Alison	v_1	**T**
111-111-1111	Peter Peterson	yes	**T**

Observe that the `grantLoan` attribute of Alice's record has been replaced by a placeholder. This value gets filled in by the assignment that defines each possible world.

This generalized form of assignments and the use of variable-valued attributes were originally proposed by Imielinski and Lipski as part of the original C-Tables formalism [21]. It has been used successfuly by several systems, most notably ORCHESTRA [18, 22] and Pip [26]. In fact, as we will discuss in Section 17, Pip [26] further generalizes this model by allowing symbolic expressions (formulas) over variables as attribute values.

[6] Note that more than two options can be modeled by multiple boolean variables. For example, 4 alternatives can be modeled with annotations $v_1 \wedge v_2$, $\neg v_1 \wedge v_2$, $v_1 \wedge \neg v_2$, and $\neg v_1 \wedge \neg v_2$, respectively.

3.1.4 U-Relations and World-Set Decompositions

Antova et. al. proposed a more backwards-compatible implementation of C-Tables called U-Relations [20]. A U-Relation is a database table that encodes a C-Table under the following restrictions: 1. No variable-valued attributes. 2. Local conditions must be pure conjunctions. 3. Atoms of local conditions must be equality comparisons between a variable and a value. To support arbitrary arbitrary boolean expressions, multiple copies of a record are allowed in the U-Relation, each annotated by a different local condition. When the full expression is needed, copies of each record are grouped and their local conditions are combined by disjunction (into disjunctive normal form).

To encode a U-Relation in a classical relational database, we first determine the maximal number of conjunctive clauses in any record. We then add twice as many integer-valued annotation fields to the record. Half are used to identify variables, while the other half identify the values.

Example 12. To encode the C-Table from Example 10, we would first see that there is at most one conjunctive atom in any local condition in the record. We add two new annotation attributes to the record: `var1` identifying the variable, and `val1` identifying the corresponding value in the equality.

Query Result (U-Relation)

SSN	name	grantLoan	var1	val1
777-777-7777	Alice Alison	no	1	1
777-777-7777	Alice Alison	yes	1	2
111-111-1111	Peter Peterson	yes	0	0

The special variable v_0 is always equal to 0, so unused fields can be filled in by the tautological expression $v_0 = 0$, as in the third row of the U-Relation.

World-Set Decompositions. Without variable-valued attributes, U-Relations introduce significant redundancy as attributes that do not vary between possible worlds are still repeated. To mitigate this redundancy, Antova et. al. propose [3] using a columnar data layout [38] in a strategy that they call World-Set Decompositions. Specifically, each record is assigned a unique identifier (e.g., a key attribute or database `ROWID`) while columns are stored independently.

Example 13. Using world set decompositions with `SSN` as a row identifier, the query result from the prior example would be decomposed into 2 separate tables:

Query Result.Name

SSN	name
777-777-7777	Alice Alison
111-111-1111	Peter Peterson

Query Result.grantLoan

SSN	grantLoan	var1	val1
777-777-7777	no	1	1
777-777-7777	yes	1	2
111-111-1111	yes	0	0

Observe that there is no uncertainty in the decomposed `Name` table, and there are no longer two copies of Alice's name being stored.

3.2 Lossy, Sampling-Based Encodings for Incomplete Data

Certain types of analysis — what are called unsafe queries [7] — can not be performed both efficiently and correctly on lossless encodings of probabilistic data. In such cases, results can be approximated by using Monte-Carlo methods. The most straightforward way to accomplish this is to select some finite set of samples $\hat{\mathcal{D}} \subset \mathcal{D}$ from the set of possible worlds; uniformly for incomplete databases or according to $P(D = \mathcal{D})$ for probabilistic databases. Analytical questions are evaluated in parallel on all possible worlds from the sample set:

$$\hat{Q}(\mathcal{D}) = \left\{ Q(D) \;\middle|\; D \in \hat{\mathcal{D}} \right\}$$

Because of the sampling process, sampling-based encodings do not distinguish between incomplete, or probabilistic databases. However, we observe that many statistical measures that might be computed over the set of results (e.g., the expectation) have no meaning for incomplete databases. We introduce two approaches to encoding sets of samples: (1) World-Annotated databases, which admit a more computationally efficient implementation using classical relational databases, and (2) Tuple Bundles, which encode uncertain data more compactly.

Note that both **certain** and **possible** are ill defined on samples. By definition the certain records are a subset of records across all samples, and the possible records are a superset.

$$\textbf{certain}(\hat{\mathcal{D}}) \subseteq \bigcap_{D \in \hat{\mathcal{D}}} D \qquad \textbf{possible}(\hat{\mathcal{D}}) \subseteq \bigcup_{D \in \hat{\mathcal{D}}} D$$

However, these are only bounds on the sets of certain and possible records.

3.2.1 World-Annotated Sample Sets

To store a set of N samples, our first, naive approach creates a single database $\mathbb{D}_{WA} \in dom(r) \times [1, N]$, annotating each record with the index of the sample it appears in. Accordingly, the sample-set is defined by de-multiplexing the records.

$$\hat{D} = \{\ \{\ r \mid \langle\ r, i\ \rangle \in \mathbb{D}_{WA}\ \}\ \mid i \in [1, N]\ \}$$

3.2.2 Tuple-Bundles

The size of $\mathbb{D}_{WA}$ is generally linear in the number of samples (i.e., $O(N)$). Unsurprisingly, the computational cost of analysis typically scales linearly as well. As with lossless encodings, eliminating redundancy can create more compact and efficient representations. One approach to eliminating redundancy is a type of record called a tuple bundle, originally proposed by Jampani et. al. [23]. We assume that a record $r = \langle\, a_1, \ldots, a_K \,\rangle$ is defined by k attribute values a_i Accordingly, a tuple bundle $\mathbf{r} = \langle\, \mathbf{a}_1, \ldots, \mathbf{a}_K, \phi \,\rangle$ is defined by a set of attributes $\mathbf{a}_j$ and a sample-vector ϕ. Each attribute may either be a single value or a vector of size N.

$$\mathbf{a}_j = \begin{cases} a_j \\ \langle\, a_{1,j}, \ldots, a_{N,j} \,\rangle \end{cases}$$

In the first case, the value a_j is constant across all samples, while the latter case defines explicit values for the attribute $a_{i,j}$ in each sample. The sample vector $\phi \in \mathbb{B}^N$ is a vector of B boolean values (bits) $\phi[i]$. Bit j being set to true (resp., false) indicates that the record is present in (resp., absent from) sample j. The corresponding sample set is defined by filtering on ϕ and plugging in attribute values.

$$\hat{D} = \{\, \{\, \langle\, a_{i,1}, \ldots a_{i,K} \,\rangle \mid (\langle\, \mathbf{a}_1, \ldots, \mathbf{a}_K, \phi \,\rangle \in \mathbb{D}_{TB}) \wedge \phi[i] \,\} \mid i \in [1, N] \,\}$$

$$\textbf{where} \quad a_{i,j} = \begin{cases} a_j & \text{if } \mathbf{a}_j = a_j \\ a_{i,j} & \text{otherwise} \end{cases}$$

4 Computing with Uncertain Tabular Data

Assume that we are given an encoding $\mathbb{D}$ (resp., $\mathbb{D}_P$) that corresponds to an incomplete (resp., probabilistic) database $\mathcal{D}$ (resp., $\langle\, \mathcal{D}, P \,\rangle$). We want to compute the answer to a question $Q(\mathcal{D})$. However, answering this question directly on $\mathcal{D}$ using possible worlds semantics is impractical, as the number of worlds is usually large. In this section we discuss techniques for computing answers more efficiently by directly manipulating the encodings $\mathbb{D}$.

4.1 Relational Algebra

Queries over tabular data are expressed through a range of different languages: SQL, SparQL, R, Spark, and others. To streamline our discussion,

we focus on a tabular data processing language called relational algebra. Relational algebra is comparatively straightforward to reason about and also captures the core data manipulation functionality of each of these other languages. Before we discuss evaluation techniques for uncertain data, we first present a short overview of normal relational algebra[7]. We then introduce strategies for evaluating relational algebra expressions over encoded uncertain databases. We follow the focus on probabilistic databases exhibited by most of the work on uncertain databases, but also note when probabilistic techniques apply to incomplete databases as well. We also focus on lossless encodings, as query evaluation over lossy encodings is a straightforward extension of classical query evaluation [23].

Operator	Notation	SQL
Table	R	`SELECT [DISTINCT] * FROM R;`
Projection	$\pi_{A,B,\ldots}(R)$	`SELECT A, B, ... FROM R;`
Selection	$\sigma_{\psi}(R)$	`SELECT * FROM R WHERE` ψ`;`
Product	$R \times S$	`SELECT * FROM R, S;`
Union	$R \cup S$	`SELECT * FROM R UNION [ALL] SELECT * FROM S;`
Aggregate	$\gamma_{A,\ldots,M \leftarrow \alpha_M,\ldots}(R)$	`SELECT A, ...,`α_M`AS M, ...` `FROM R GROUP BY A, ...;`

Fig. 3: Relational algebra

Relational algebra concerns itself with sets (resp., bags) of records called tables (R, S, T, ...). An individual record $r \in R$ is a set of attribute/value pairs $r = \langle A : v_A, B : v_B, \ldots \rangle$, and we assume that all records in a table have identical sets of attributes. We refer to this set of attributes as the table's schema $sch(R)$. Relational algebra, as we use it, defines 6 operators, summarized in Figure 3: Input Tables, Projection, Selection, Product, Union, and Aggregate. Apart from the table operator, each operator takes one or more other operators as input and produces an output that may be saved as a table or passed to another operator. Hence, operators can be linked together to express complex computations.

Projection transforms each record of its input, producing records with attributes given by a set of target columns $(A, B, ...)$. When working with sets, projection also ensures that the output is free of duplicates.

Selection filters its input down to records that satisfy the condition ψ.

Product pairs every record in one of its inputs with every record in the other. The combination of Selection and Product operators ($\sigma_{\psi}(R \times S)$), is often called a Join ($R \bowtie_{\psi} S$]).

Union merges records from two input tables. If working with sets, Union also ensures that there are no duplicates in the inputs.

[7] For a more thorough introduction, we refer the interested reader to a textbook by Garcia-Molina et. al. [14]

(Group-By) Aggregate creates groups of records according to a list of attributes $(A, B, \ldots)$. Records in each group are summarized by one or more aggregate functions ($\alpha_i \in \{\texttt{SUM}, \texttt{COUNT}, \texttt{AVG}, \texttt{MIN}, \ldots\}$), and one record per group is output. If no grouping attributes are given, aggregation treats its entire input as a single group.

4.2 Extensional Evaluation

We first consider the Tuple-Independent model [4, 40, 39]. Recall that a Tuple-Independent probabilistic database is encoded by annotating each record with an extra field $\phi \in [0, 1]$ denoting the independent probability of this record being in any given possible world. Naively, we might try modifying relational algebra operators to preserve these annotations, a strategy called "Extensional" evaluation [40, 39] of relational algebra. That is, for each operator, we define an evaluation strategy that ensures that each output row is annotated with the independent probability of the result. Figure 4 illustrates this strategy for queries evaluated over sets. We next discuss these operators — we omit Aggregation for the moment.

Projection. For projection, we eliminate duplicate rows using a group-by query. Each resulting record exists if any records that share projection attributes exist. Assuming that each row in the input is independent, the corresponding probability is a disjunction of independent events $(1 - (1 - p(t_1)) \cdot (1 - p(t_2)) \cdot \ldots)$

Selection. Selection has no impact on probabilities of records. Records that are filtered out are excluded from the result regardless of their probability. Records that are not filtered out appear in the result with their original probabilities.

Product. For a row to appear in the output of a product, it must have resulted from one row in each of the product's inputs. Assuming that the inputs are independent, the probability of each output row can be computed as a conjunction of 2 independent events $(p(t_1) \cdot p(t_2))$.

Union. Union itself does not affect the probability of its inputs. However, during duplicate removal union may need to merge record probabilities. It does this in the same way as during duplicate removal for projection.

Modifying a relational algebra expression to maintain the probability annotation attribute ϕ through extensional evaluation adds minimal computational overhead. However, extensional evaluation has several serious limitations that all stem from its use of the Tuple-Independent model to represent state in between operators. First, the Tuple-Independent model can not efficiently represent the outputs of the aggregate operator: The size of the output grows exponentially with the size of the input. Second, even if the input to

Original Operator	Probabilistic Implementation
$\pi_{A,B,\ldots}(R)$	`SELECT A, B, ..., 1 - PROD(1-`ϕ`) FROM R GROUP BY A, B, ...;`
$\sigma_{\psi}(R)$	`SELECT * FROM R WHERE` ψ`;`
$R \times S$	`SELECT *, R.`ϕ` * S.`ϕ` AS` ϕ `FROM R, S;`
$R \cup S$	`SELECT *, 1 - PROD(1-`ϕ`) FROM ( SELECT * FROM R UNION SELECT * FROM S ) GROUP BY *;`

Fig. 4: Extensional evaluation implemented in SQL

a relational algebra expression is independent, the expression may introduce correlations between rows of output.

Example 14. Continuing our running example, consider the loan approval table and another table of homes available for purchase scraped from websites and government records. Scraping is an imprecise process, and the record at *45 Bassett* may not actually represent a home available for purchase.

possible(`forSale`)

address	price	ϕ
123 Acacia	200k	1.0
45 Bassett	150k	0.9

possible($Q(\mathcal{D})$ $\times$ `forSale`)

SSN	name	grantLoan	address	price	ϕ
777-777-7777	Alice Alison	yes	123 Acacia	200k	0.4
777-777-7777	Alice Alison	yes	45 Bassett	150k	0.36
777-777-7777	Alice Alison	no	123 Acacia	200k	0.6
777-777-7777	Alice Alison	no	45 Bassett	150k	0.54
111-111-1111	Peter Peterson	yes	123 Acacia	200k	1.0
111-111-1111	Peter Peterson	yes	45 Bassett	150k	0.9

Consider the product of this new table with the result table from our running example (obtained via Extensional evaluation). There are two types of correlations in the result records. The presence of the second, fourth, and sixth possible records depends on whether or not the record for *45 Bassett* is present in `forSale`. Meanwhile the first two records must always appear together, and are mutually exclusive with the third and fourth result records. The records are not independent, and the measure ϕ annotating each record can no longer be used to compute the probability of worlds containing the record. Note however, that for this example the probabilities annotating each record do correspond to the *marginal* probability of that record being in the result.

Relational algebra introduces correlations, and program outputs are not guaranteed to be tuple-independent. Thus, it is possible that the resulting probability annotations will be meaningless. Fortunately that is not always the case. Dalvi and Suciu identified a particular class of relational algebra

expressions termed "safe" [39, 7], as well as a procedure for (1) rewriting expressions into equivalent safe forms, or (2) determining that there is no equivalent safe expression. When a safe expression is evaluated using the extensional rules, every output record will be annotated with the record's confidence (marginal probability of being in the result). Extensional evaluation also has value for unsafe expressions. As shown by Gatterbauer and Suciu [15], extensional evaluation can be used to establish bounds on the actual confidence values.

4.3 Intensional Evaluation

For computing with unsafe relational algebra expressions, we need an evaluation strategy that takes into account potential inter-row correlations. Our next approach, called "Intensional" evaluation, uses C-Tables as an underlying data representation. Recall that records in C-Tables are annotated with a boolean formula ($f_r(v_1, \ldots, v_N)$), parameterized by variable symbols (v_i). Possible worlds are defined by assignments of values to variables; Records are included in a result in worlds that assign values that satisfy the boolean expression.

Intensional query evaluation [39, 21] is closely related [17] to the provenance (i.e., lineage or pedigree) of query answers. Under intentional evaluation, each operator annotates output records with the conditions that need to hold on the assignment for the record to be in the result. Hence, correlations are explicitly captured in the query results as variables that appear repeatedly in a formula or across the annotations of multiple records.

Original Operator	**Probabilistic Implementation**
$\pi_{A,B,\ldots}(R)$	`SELECT A, B, ..., BOOLEAN_OR(`ϕ`) FROM R GROUP BY A, B, ...;`
$\sigma_{\psi}(R)$	`SELECT * FROM R WHERE` ψ`;`
$R \times S$	`SELECT *, BOOLEAN_AND(R.`ϕ`, S.`ϕ`) AS` ϕ `FROM R, S;`

$R \cup S$	`SELECT *, BOOLEAN_OR(`ϕ`) FROM (`
	`  SELECT * FROM R UNION SELECT * FROM S`
	`) GROUP BY *;`

Fig. 5: Intensional evaluation implemented in SQL

For Intensional evaluation (a possible implementation is shown in Figure 5), operators follow a virtually identical pattern to extensional evaluation, except that the resulting annotation ϕ is a boolean formula. In the probabilistic database literature, this type of formula is often called Lineage. Once the result is computed, the problem of computing marginal probabilities be-

comes one of simple inference. For each tuple, we are given boolean formula and a distribution over binary variables appearing in the formula. We need to compute the marginal probability of this boolean formula being true [8].

The problem of inference has been well studied in the general context [28]. The specific problem of computing marginals under constraints belongs to the general problem of counting solutions to boolean formulas. Exact solutions are exponential in the size of the input (complexity class #P [33]) and numerous approximation schemes have been developed. However, probabilistic databases admit several specializations of general techniques. We next discuss several of these.

KLM Estimators. It is well known that any relational algebra expression which exclusively uses the operators we have introduced here can be rewritten into a normal form which consists of a *union of conjunctive queries* (or *UCQ* for short). A *conjunctive query* (*CQ*), is a query without union which consists of a projection over the result of a selections which in turn is applied to the result of zero or more cross-products. After such a rewriting, it is trivial to see that boolean formulas annotating results are guaranteed to be in disjunctive normal form, because the final union will connect the formulas produced by individual CQs through OR while the formulas produced by a CQ are conjunctions (hence the name). As observed by Olteanu et. al. [32], this makes a form of gibbs sampling proposed by Karp, Luby, and Madras [24] ideal for probabilistic databases. The KLM scheme begins with a disjunction $C_1 \vee C_2 \vee \ldots \vee C_N$ of conjunctive clauses C_i. It initially assumes that each conjunctive clause is disjoint:

$$p(C_1 \vee C_2 \vee \ldots \vee C_N) = p(C_1) + p(C_2) + \ldots + p(C_N)$$

This assumption is an over-estimate, as variable assignments that satisfy 2 (or more) of the conjunctive clauses are counted twice (or more). The scheme then attempts to derive a corrective factor by repeatedly sampling clauses at random, and computing the expected number of clauses that a satisfying assignment for the clause would also satisfy. An approach by Dagum et. al. [6] improves on this approach by bounding the number of samples required to estimate record confidence within desired $\epsilon - \delta$ bounds.

Anytime Approximation. Another distinction to be found in this setting, is that data analytics are often interactive processes. The process of approximating confidence values can also be made interactive, allowing the analyst to decide on-the-fly when a result is "accurate enough" before terminating the process. One such approach, proposed by Olteanu et. al. [13], alternates between using an approximate estimator like KLM, and repeated refinement of the boolean formula towards an one consisting exclusively of disjoint clauses through Shannon expansion. Given enough time, this approach eventually converges to an exact value for a record's confidence.

[8] Observe that a binary version of this problem can be applied in the case of incomplete databases. A tuple is certain if its local condition is implied by the global condition.

Top-K Estimation. One particular specialization of probabilistic databases that is of interest is finding the most likely records in the output of a relational algebra expression [30, 35] For example, given uncertain inputs describing existing findings of protein-protein interactions, we might wish to predict other likely interactions [10]. In general, this problem can be framed as the task of finding the K most probable records (the Top-K records) from the result. In this case, our computational job is easier, as exact probabilities are not required. We only need a sufficient approximation of each probability to decide whether the record belongs in the Top-K or not. One family of approaches proposes using early cutoffs in approximations [35, 37]. A related approach by Gatterbauer and Suciu uses intensional evaluation to establish bounds on the probability of a record [15], allowing for early cutoffs as well. A final approach by Li et. al. [30] attacks a further specialization aiming at the "Best" results. Here, the notion of "best" is formalized by one of several different strategies for combining a user-provided ranking function over result records with the probability of records being in the result.

Attribute-Level Uncertainty. Thus far, most of our discussions have centered around record-level uncertainty: the presence of a specific record with a specific set of attribute/value pairs in the result set. However in many situations it is not the record that is uncertain, but rather one of the values of an attribute of that record. For example, when computing an aggregate over an incomplete or probabilistic table, aggregate values are likely to be uncertain, while the groups to which they belong need not be. Although most work on probabilistic databases focuses on record-level uncertainty, several efforts have attempted to encode uncertainty appearing in attributes. The original Imielinski and Lipski formalization of C-Tables [21] allows variable symbols to appear in place of values in tabular data — variable assignments with non-boolean values as well are used to assign values to these variables. Notably, this means that selection can modify the formula annotating selected records. Singh et. al. [36] allows attributes to take values defined by normal distributions. Antova et. al. [3] propose a strategy that fragments records in tables, replacing the table with a set of tables, one per field. Finally, Kennedy et. al. [25] builds on the C-Tables formulation to construct formulas for the values of uncertain attribute fields, which can be evaluated after variables are replaced by an assignment.

4.4 Virtual C-Tables

A recently proposed evaluation strategy based on C-Tables instead *virtualizes* uncertainty. This approach uses a generalization of C-Tables called *Virtual C-Tables* (or *VC-Tables* for short). We first introduce this generalization and then explain the evaluation strategy that utilizes it.

Recall that in a C-Table, the attribute values of a tuple are constants or variables from a set $\mathbb{V}$ and the local condition of a tuple is a boolean formula over comparisons between variables and constants. While C-Tables are quite powerfule, there still exist operators whose result are complicated or impossible to express as C-tables.

Example 15. Assume we want to require customers to pay an application fee for every loan application they make that is 1% of the requested loan amount and that we want to automatically accept loan applications if this fee is higher than $10,000. Using the applications table (attributes SSN and loanAmount), we can determine the set of loans that will be automatically approved as follows:

```
SELECT SSN, loanAmount, loanAmount * 0.01 AS fee
FROM applications
WHERE loanAmount * 0.01 > 10000.0;
```

Consider the C-Table applications shown below where we do not know what the loan amount for the customer with SSN 111-777-2222 which is encoded by setting the value of attribute loanAmount for this record to a variable, say v_1.

SSN	loanAmount	ϕ
777-777-7777	200,000	**T**
111-777-2222	v_1	**T**

The loan for the customer with SSN 111-777-2222 will be automatically approved if $v_1 * 0.01$ (the fee) is larger than $10,000. In this case the fee attribute of the result record has to be set of $v_1 * 0.01$. This correlation between the loanAmount and fee attribute is hard to express in a standard C-Table since there is no support for arithmetic operations. In fact, it can only be represented if the domain of the loanAmount is finite. In this case, we would have to represent each possible loanAmount and fee pair that fulfills the condition as a separate tuple. Some of these tuples are shown below.

SSN	loanAmount	fee	ϕ
111-777-2222	1,000,001	10,000.01	$v_1 = 1,000,001$
111-777-2222	1,000,002	10,000.02	$v_1 = 1,000,002$
111-777-2222	1,000,003	10,000.03	$v_1 = 1,000,003$
...	...	...	...

VC-Tables overcome this limitation of C-Tables by allowing attribute values and inputs to comparions in local and global conditions to be symbolic expressions using arithmetic operators, conditionals (if-then-else), constants, and variables. Possible worlds are still defined over variable assignments. The only difference is that the attribute values of a tuple in a possible world are determined by evaluating the symbolic expressions of the tuple under the assignment α corresponding to the possible world. For details of the formal definition of these expressions see [41, 26].

Example 16. Continuing with the previous example, we can compactly represent the query result as a VC-Table as follows:

SSN	loanAmount	fee	ϕ
111-777-2222	v_1	$v_1 * 0.01$	$v_1 * 0.01 > 10000$

For example, for the assignment $v_1 = 1,500,000$ we get the possible world:

SSN	loanAmount	fee
111-777-2222	1,500,000	15,000

We can extend VC-Tables to support probabilistic databases by defining a probability distribution over possible variable assignments.

Having introduced VC-Tables, we now explain how the Mimir PQP middleware [41, 31] uses such encodings to virtualizes PQP. Mimir rewrites probabilistic queries into deterministic queries over a deterministic encoding of uncertain data such that the rewritten queries faithfully preserve the semantics of the probabilistic queries. To accomplish this, Mimir uses an extended form of relational algebra called *Variable-Generating Relational Algebra*, or *VG-RA* [25].

Using VG-RA, uncertainty is introduced into deterministic data through queries. A VG-RA expression defines VC-Tables by allowing expressions to generate variable symbols through special functions called Variable-Generating or VG-Terms. A VG-Term denoted $VG(\cdot)$ can appear in any boolean or arithmetic expression in any Project, Select, or Aggregate operator in a VG-RA query. The input of a VG-Term controls the name of the variable generated by VG-RA for a given input record. For instance, a VG-Term $VG(\mathsf{name})$ would return a unique variable for each name attribute value from the input of the operator where the VG-Term is used. The result of a VG-RA expression is an incomplete database — a VC-Table. Essentially, once a variable is introduced by a VG-Term, expressions involving this VG-Term are evaluated symbolically. VG-RA allows the generated variables to be associated with a (potentially joint) distribution over possible assignments (we do not show the language constructs for this here).

Example 17. Recall the `customers` table from Figure 1. Although this table is missing several values, there is no uncertainty about this fact. Mimir allows users to create a cleaned "view" over the data; For the `customers` table, Mimir would use the following query:

```
SELECT SSN, name, assets, numMortgages,
       CASE WHEN income IS NULL THEN VG('income', RID)
            ELSE income END AS income,
       CASE WHEN ownsProperty IS NULL THEN VG('ownsProperty', RID)
            ELSE ownsProperty END AS ownsProperty
FROM customers;
```

For each of the two attributes with missing values, Mimir replaces `NULL` by using the SQL fragment

```
CASE WHEN x IS NULL THEN VG('x', RID) ELSE x
```

If the value is present, the fragment leaves it unchanged. If it is `NULL`, the fragment replaces it with a variable created by the VG Term `VG('x', RID)`, where `RID` uniquely identifies each row of the table. By being keyed on the attribute name, as well as a unique row identifier, one fresh variable is instantiated for every column and row. Independently, Mimir trains a model on the same data, uses interpolation, or any other imputation technique. The resulting models are then linked to the new variables. For instance, omitting the probability distributions over assignments, the result of the query shown above over the database from Figure 1 (assuming an additional attribute RID as supported by many database systems) would be:

RID	**SSN**	**name**	**income**	**assets**	**numM**	**ownsP**	ϕ
1	777-777-7777	Alice Alison	\$60,000	\$200,000	0	$v_{\prime ownsProperty\prime,1}$	**T**
2	333-333-3333	Bob Bobsen	\$102,000	\$15,000	0	no	**T**
3	111-111-1111	Peter Petersen	$v_{\prime income\prime,3}$	\$90,000	1	yes	**T**
4	555-555-5555	Arno Arnoldson	\$95,000	\$30,000	0	yes	**T**

c

In most probabilistic databases, queries are assumed to be deterministic and the data is non-deterministic (i.e., $Q(\mathcal{D})$). Conversely, in Mimir the reverse is true, as all uncertainty is introduced through VG-Terms (i.e., as part of $\mathcal{Q}(D)$). In addition to the other benefits in terms of compact representation of the result of arithmetic expression brought by VC-Tables, this allows a evaluation strategy to be chosen at query time. This is important in practice since different evaluation strategies and approximations may exhibit vastly different performance and performance may be affected significantly by the structure of the query that is evaluated. Thus, by allowing the strategy to be chosen per query is critical to trade performance against accuracy of the result using the techniques introduced in this section and Section 5.

5 Presenting Uncertain Tabular Data

We next survey techniques for presenting uncertain tabular data to users [29], develop a taxonomy of presentation strategies, and relate these strategies to algorithms for computation over uncertain data. That is, we discuss techniques that allow us to communicate to a user the set (resp., distribution) of possible worlds represented by an incomplete database $\mathcal{D}$ (resp., probabilistic database $\langle\, \mathcal{D}, P \,\rangle$) or an encoding $\mathbb{D}$ thereof (resp., $\mathbb{D}_P$).

5.1 Tuple Identity

We have introduced two forms of uncertainty: Record-level uncertainty (is a record part of a result or not), and Attribute-level uncertainty (what is the value of a specific attribute). There is a tension between these two forms of uncertainty: At what point are the attributes of two records from different possible worlds sufficiently similar to be considered the same record? If they are considered the same record, then (with respect to the two possible worlds) we have one certain record with uncertain attributes. Conversely, if the records are different, then we have two uncertain records, with no uncertainty about their attributes.

To resolve this tension, different uncertainty management systems define — often indirectly — a record identity function $id(r)$. $id(r)$ assigns to every record an identifier that is unique within a possible world of the database $\mathcal{D}$. The primary role of identifiers is to gather instances of the same records from different possible worlds, while allowing the precise definition of record "sameness" to vary based on the needs of the representation. Record identifiers allow us to define, for a particular possible world $D \in \mathcal{D}$, the set of identifiers appearing in the possible world. We refer to the set of possible worlds that contain a tuple with the same identifier as a tuple r as the support of r and denote it as $sup(r, \mathcal{D})$.

$$ids(D) = \{\ id(r)\ \mid r \in D\ \} \qquad sup(r, \mathcal{D}) = \{\ D\ \mid D \in \mathcal{D} \wedge id(r) \in ids(D)\ \}$$

The support of r, in turn, gives us definitions for possible and certain records.

$$r \textbf{ is certainly in } \mathcal{D} := \left[\ sup(r, \mathcal{D}) = \mathcal{D}\ \right]$$

$$r \textbf{ is possibly in } \mathcal{D} := \left[\ |sup(r, \mathcal{D})| \geq 1\ \right]$$

Likewise, we can define the set of possible values of a record r's attribute A as the set of values of A in all records with the same identity.

$$\textbf{possible}(r[A] \in \mathcal{D}) := \{\ r'[A]\ \mid id(r) = id(r') \wedge r' \in D \wedge D \in \mathcal{D}\ \}$$

This in turn allows us to say that a record r's attribute A is certain if and only if it has exactly one possible value, or that it is bounded if its possible values satisfy some constraint.

$$r[A] \textbf{ is certain in } \mathcal{D} := \left[\ \left|\textbf{possible}(r[A] \in \mathcal{D})\right| = 1\ \right]$$

$$r[A] \textbf{ is bounded by } [\ell, h] \textbf{ in } \mathcal{D} := \forall a \in \textbf{possible}(r[A] \in \mathcal{D}) : \ell \leq a \leq h$$

If the set of possible worlds has an associated probability measure $p(\mathcal{D})$, we can define the confidence of a record as the marginal probability over all possible worlds with a record with the same identifier.

$$conf(r \in \mathcal{D}) := \sum_{D \in sup(r)} p(D)$$

We call any subset of records $S \subseteq \bigcup_{D \in \mathcal{D}} D$ a *summary* if no pair of records in S share an id. We call a summary *complete* if every identifier in $\mathcal{D}$ is represented. That is, S is a complete summary of $\mathcal{D}$ if and only if:

$$S \subseteq \bigcup_{D \in \mathcal{D}} D \quad \wedge \quad \forall r_1 \neq r_2 \in S : id(r_1) \neq id(r_2) \quad \wedge \quad ids(S) = \bigcup_{D \in \mathcal{D}} ids(D)$$

5.2 Compact Encodings of Possible Worlds

With this terminology in place, we are now ready to describe the space of the representational schemes used by existing uncertain and probabilistic database systems. A specific representation is the result of three categories of representational features. (1) **Equivalence**, or how the scheme decides which tuples to place in a specific tuple group, (2) **Filtering**, or what subset of the complete summary relation the scheme incorporates, and (3) **Statistics**, or how the scheme summarizes properties of each tuple group. When appropriate, we also distinguish between **tuple-level**, and **attribute-level** statistics. Figure 6 illustrates how existing PQP schemes relate to these features.

Alg. Family	Example Systems	Equivalence	Filtering	Statistics
Monte Carlo	MCDB [23], Jigsaw [27]	Implicit	Samples	Agg, Conf.
Chase	Data Exchange [11]	Set	Certain	None
Local Condition	MayBMS [3, 20], Orchestra [18, 22]	Set	Possible	Enum, Conf.
Pruning	Top-K [37], Dissociation [15]	Set	Top-K Post.	None
PC-Tables	PIP [26], Orion [36]	Implicit	Possible	Agg, Enum, Conf
VC-Tables	Mimir [41, 31]	Implicit	Best Guess	Taint, Top-K Prior
Model	Velox [5], MauveDB [9]	Implicit	Possible	Conf

Fig. 6: PQP systems in terms of the representational semantics used to communicate uncertain tables.

5.2.1 Record Equivalence: Assigning Identifiers

Record identifiers eliminate redundancy in the summary by allowing us to represent certain forms of conflicts through attribute-level uncertainty. To date, existing probabilistic and incomplete database systems have adopted

one of two approaches to identifiers that we call Set- and Implicit-identity. The vast majority of literature on probabilistic databases (e.g., [37, 32, 4, 13, 39, 7, 15, 16] ignores attribute level uncertainty. In this approach, which we term Set-identity, the entire record is used as an identifier. Under Set-identity, two records have the same identity if and only if the values of their attributes are identical.

Conversely, systems [26, 23, 36, 40, 41] that support attribute-level uncertainty typically give each tuple an implicit identifier. In this approach, which we term Implicit identity, each record in input tables is assigned a unique identifier (analogous to the `ROWID`s of popular database systems). This identifier is propagated through relational algebra expressions in a manner that mimics database provenance [17] (i.e., lineage or pedigree) as illustrated in Figure 7. Projection and selection preserve the identity of a record. Product deterministically derives a new identifier for each output record from the identifiers of the records used to derive it. Union deterministically derives new identifiers for each output record based on the input record's identifiers, and which side of the union it came from [9]. Aggregates produce entirely new records. We identify the resulting records by deriving an identifier from the grouping attributes, or using a default attribute if there are no grouping attributes.

Example 18. Consider the `customers` table in the two possible worlds D_1 and D_2 of Figure 2. Set-identity schemes (e.g., u-relations [2] or semiring annotations [16]) assign identity based on record values. The records for Bob, Peter, and Arno are each assigned one identifier across both worlds, while each of the two possible records for Alice is assigned its own identifier. Accordingly, records for Bob, Peter, and Arno are certain, while the other records are only possible.

Conversely, implicit-identity schemes (e.g., MCDB [23], Pip [26], or Mimir [41, 31]) might use a key attribute like `SSN` as an identifier for the row. In such schemes, all four records are certain and only Alice's `ownsProperty` attribute is uncertain.

5.2.2 Filtering Uncertain Records

It is often helpful to further summarize possible relations by filtering out low-importance record identities. The most general approach to filtering is based on a record's support. Most incomplete database systems intended for Data Exchange, Cleaning, or Fusion (e.g., [11]) return only certain tuples of a query result. Conversely, many probabilistic database systems present the complete set of possible results [2].

Certain and possible results represent two extremes of a spectrum. The former may omit potentially valuable information, while the latter might

[9] This prevents repeated identifiers if a record appears on both sides

Operator	Notation	SQL
Table	R	`SELECT *, ROWID FROM R;`
Projection	$\pi_{A,B,\ldots}(R)$	`SELECT A, B, ..., ROWID FROM R;`
Selection	$\sigma_{\psi}(R)$	`SELECT * FROM R WHERE` ψ`;`
Product	$R \times S$	`SELECT *, MERGE_IDS(R.ROWID, S.ROWID) AS ROWID FROM R, S;`
Union	$R \cup S$	`SELECT *, MERGE_IDS(R.ROWID, '1') AS ROWID FROM R UNION [ALL] SELECT *, MERGE_IDS(S.ROWID, '2') AS ROWID FROM S;`
Aggregate	$\gamma_{A,\ldots,M \leftarrow \alpha_M,\ldots}(R)$	`SELECT A, ...,`α_M`AS M, ..., MERGE_IDS(A, B, ...) AS ROWID FROM R GROUP BY A, B, ...;`

Fig. 7: Deriving implicit row identifiers

produce many records, overwhelming the user. The search for an intermediate 'sweet spot' has led to the emergence of a variety of semantics that is a superset of certain answers and a subset of possible answers. (1) **sample** filtering [23, 27], includes a union of all records from a (lossy) sampled set of possible worlds. (2) **threshold** filtering includes all records with a support or confidence that larger than a threshold size or probability. (3) **top-k prior** filtering [41, 31] includes all records that appear in the results for the k most likely *possible worlds*. We call the special case of the top-1 prior **best guess** filtering, as these are the results from the most likely possible world. Finally, (4) **top-k posterior** results include only the top-k records, as ranked by the confidence or support of the *result itself* [10, 30].

Note the distinction between the two top-k filtering strategies: top-k prior filters before marginalizing, while top-k posterior filters afterwards. The top-k posterior filtering strategy is particularly appropriate for settings where the user is searching for the most likely result. For example a user examining a medical diagnostic query result is probably interested in the most likely diagnosis. We note that although this is appropriate for such specialized use cases like diagnostic or recommender systems, such representations can lead to a confusing proliferation of semantics [30].

5.2.3 Statistics for Uncertain Attributes

The final point to consider when designing a summary representation is how to represent the records themselves. For this, we need to convey both record- and attribute-level uncertainty.

Record-level statistics. Uncertain result records do not appear in all possible worlds. When communicating this information to the user, we effectively wish to communicate some features of the record's support. Although some schemes are capable of **enum**erating the set of all worlds that a record appears in, this capability is typically expensive. Rather, most PQP schemes compute or approximate a record's **conf**idence[7, 13, 15]. A simpler approach

uses **taint** annotations [41, 31, 29] to differentiate between certain and possible records.

Attribute-level statistics. Unless set-identity is being used, records may have uncertain attributes as well. Most probabilistic and incomplete database schemes assume that we are only interested in summarizing individual attributes and not more general properties. The most common strategy is to construct **agg**regate summaries of the attribute's values across the record. Histograms [23], expectations [26, 23], confidence bounds [27], or hard bounds have been used as aggregate summaries. Another approach is to summarize an attribute of a record by one, or a set of possible values [41, 10]. Any record filtering strategies can be leveraged to decide which possible values to include, e.g., top-1 prior [41] or top-k posterior [10, 37].

6 Conclusions

In this paper, we explain how uncertainty arises in detection and resolving of data quality issues, and how this uncertainty may cause data quality issues in analysis results which often remain undetected. Such untrustworthy analysis results can in turn have severe adverse real world effects such as unfounded scientific discoveries, financial damages, or even affect people's physical well-being (e.g., medical decisions based on data with low quality). The main purpose of this work is to give an overview of uncertain datamanagement techniques and raise awareness of how these techniques can be applied to explain how heuristic resolutions to data quality problems affect the quality and trustworthiness of analysis results.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. L. Antova and C. Koch. On apis for probabilistic databases. In *QDB/MUD*, pages 41–56, 2008.
3. L. Antova, C. Koch, and D. Olteanu. Maybms: Managing incomplete information with probabilistic world-set decompositions. In *ICDE*, pages 1479–1480. IEEE Computer Society, 2007.
4. J. Boulos, N. N. Dalvi, B. Mandhani, S. Mathur, C. Ré, and D. Suciu. MYSTIQ: a system for finding more answers by using probabilities. In *SIGMOD Conference*, pages 891–893. ACM, 2005.
5. D. Crankshaw, P. Bailis, J. E. Gonzalez, H. Li, Z. Zhang, M. J. Franklin, A. Ghodsi, and M. I. Jordan. The missing piece in complex analytics: Low latency, scalable model management and serving with velox. In *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*. www.cidrdb.org, 2015.

6. P. Dagum, R. M. Karp, M. Luby, and S. M. Ross. An optimal algorithm for monte carlo estimation. *SIAM J. Comput.*, 29(5):1484–1496, 2000.
7. N. Dalvi and D. Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6):30:1–30:87, jan, 2013.
8. G. V. den Broeck and D. Suciu. Query processing on probabilistic data: A survey. *Foundations and Trends in Databases*, 7(3-4):197–341, 2017.
9. A. Deshpande and S. Madden. Mauvedb: Supporting model-based user views in database systems. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 73–84, New York, NY, USA, 2006. ACM.
10. L. Detwiler, W. Gatterbauer, B. Louie, D. Suciu, and P. Tarczy-Hornoch. Integrating and ranking uncertain scientific data. In *ICDE*, pages 1235–1238, 2009.
11. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89 – 124, 2005. Database Theory.
12. W. Fan. Dependencies revisited for improving data quality. In M. Lenzerini and D. Lembo, editors, *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2008, June 9-11, 2008, Vancouver, BC, Canada*, pages 159–170. ACM, 2008.
13. R. Fink, J. Huang, and D. Olteanu. Anytime approximation in probabilistic databases. *VLDB J.*, 22(6):823–848, 2013.
14. M. Garcia, J. Ulman, and J. Wisdom. Database systems: the complete book, 2002.
15. W. Gatterbauer and D. Suciu. Dissociation and propagation for approximate lifted inference with standard relational database management systems. *VLDBJ*, 26(1):5–30, 2017.
16. T. Green and V. Tannen. Models for incomplete and probabilistic information. In T. Grust, H. Höpfner, A. Illarramendi, S. Jablonski, M. Mesiti, S. Müller, P.-L. Patranjan, K.-U. Sattler, M. Spiliopoulou, and J. Wijsen, editors, *Current Trends in Database Technology – EDBT 2006*, volume 4254 of *Lecture Notes in Computer Science*, pages 278–296. Springer Berlin Heidelberg, 2006.
17. T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.
18. T. J. Green, G. Karvounarakis, N. E. Taylor, O. Biton, Z. G. Ives, and V. Tannen. ORCHESTRA: Facilitating collaborative data sharing. In *SIGMOD*, pages 1131–1133, 2007.
19. P. Guagliardo and L. Libkin. Making SQL queries correct on incomplete databases: A feasibility study. In *PODS*, pages 211–223. ACM, 2016.
20. J. Huang, L. Antova, C. Koch, and D. Olteanu. MayBMS: A probabilistic database management system. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 1071–1074, New York, NY, USA, 2009. ACM.
21. T. Imieliński and W. Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, sep, 1984.
22. Z. G. Ives, T. J. Green, G. Karvounarakis, N. E. Taylor, V. Tannen, P. P. Talukdar, M. Jacob, and F. Pereira. The orchestra collaborative data sharing system. *SIGMOD Rec.*, 37(3):26–32, sep, 2008.
23. R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas. MCDB: A monte carlo approach to managing uncertain data. In *SIGMOD*, pages 687–700, 2008.
24. R. M. Karp and M. Luby. Monte-carlo algorithms for enumeration and reliability problems. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, 0:56–64, 1983.
25. O. Kennedy. The PIP MayBMS plugin. http://maybms.sourceforge.net.
26. O. Kennedy and C. Koch. PIP: A database system for great and small expectations. In *ICDE*, pages 157–168. IEEE Computer Society, 2010.

27. O. A. Kennedy and S. Nath. Jigsaw: Efficient optimization over uncertain enterprise data. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 829–840, New York, NY, USA, 2011. ACM.
28. D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.
29. P. Kumari, S. Achmiz, and O. Kennedy. Communicating data quality in on-demand curation. In *QDB*, 2016.
30. J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probabilistic databases. *pVLDB*, 2(1):502–513, aug, 2009.
31. A. Nandi, Y. Yang, O. Kennedy, B. Glavic, R. Fehling, Z. H. Liu, and D. Gawlick. Mimir: Bringing ctables into practice. Technical report, ArXiv, 2016.
32. D. Olteanu, J. Huang, and C. Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, pages 145–156. IEEE Computer Society, 2010.
33. C. H. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.
34. S. Parsons. *Probabilistic Graphical Models: Principles and Techniques* by daphne koller and nir friedman, MIT press, 1231 pp., $95.00, ISBN 0-262-01319-3. *Knowledge Eng. Review*, 26(2):237–238, 2011.
35. C. Ré, N. N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895. IEEE Computer Society, 2007.
36. S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. Hambrusch, and R. Shah. Orion 2.0: Native support for uncertain data. In *SIGMOD*, pages 1239–1242, 2008.
37. M. A. Soliman, I. F. Ilyas, and K. C. Chang. Top-k query processing in uncertain databases. In *ICDE*, pages 896–905. IEEE Computer Society, 2007.
38. M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik. C-store: A column-oriented DBMS. In *VLDB*, pages 553–564. ACM, 2005.
39. D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.
40. J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. *Technical Report*, 2004.
41. Y. Yang, N. Meneghetti, R. Fehling, Z. H. Liu, and O. Kennedy. Lenses: An on-demand approach to etl. *Proc. VLDB Endow.*, 8(12), 2015.