Uncertainty in the Last Mile

Poonam Kumari[β], Ying Yang[β], Lisa Lu[β], Arindam Nandi[Hβ], Dieter Gawlick[O],
Zhen Hua Liu[O], Beda Hammerschmidt[O], Ronny Fehling[A], Boris Glavic[I], Oliver Kennedy[β]
β: University at Buffalo, O: Oracle, H: HPE/Vertica, A: Airbus, I: Illinois Inst. Tech.
{poonamku, yyang25, lisalu, okennedy}@buffalo.edu arindam.nandi@hpe.com
{dieter.gawlick, zhen.liu, beda.hammerschmidt}@oracle.com ronny.fehling@airbus.com bglavic@iit.edu

***Abstract*—Over the past decade, there has been a proliferation of low-veracity data sources like personal sensing and data extracted from the web. Decades of research on probabilistic data management have explored how to make uncertainty a first-class primitive in database management systems. However, efforts in this space have remained largely algorithmic, focusing more on how to efficiently perform computations over probabilistic or incomplete data and less on how the data is to be consumed. In this paper, we tackle the problem of uncertain data management from a user-focused perspective. We conducted a usability study in which we explored how users interact with different representations of uncertainty. As one result of this study, we found that low-detail representations of uncertainty could still be *effective* at communicating uncertainty and could help users to *efficiently* make decisions based on uncertain data. We then identified a range of query evaluation strategies well suited for generating these representations and implemented them in our probabilistic data cleaning system, Mimir. Unsurprisingly, we found a clear tradeoff between the level of detail conveyed by the representation and the computational cost of generating it. In summary, this paper demonstrates that, for user-facing applications, existing techniques developed for probabilistic data management can be simplified, and that it is possible to create a user-facing probabilistic database competitive with classical DBMSes.**

I. Introduction

Uncertainty is increasingly relevant to all facets of data management, from small-scale small personal sensing applications to large corporate or scientific data analytics. In these and many other settings, heuristics play an active role in enforcing data quality, and in doing so decide what information the end-user is exposed to. As a simple example, several modern calendar and address book programs populate themselves from a linked email account. The program's heuristics decide which sections of text describe an event or contact. The result of these heuristics is a table of (typically relational) data that programs can then display to the user. Unfortunately, the crisp precision of a table of query results — or even just a collection of numbers and names — can conceal potential unreliability of the values shown. In the more extreme cases, presenting this information as if it were valid can ruin lives. Examples include credit agencies[1] and the TSA no-fly list[2], both of which frequently contend with automation-related issues ranging from entity resolution to missing or untrustworthy data.

Historically, the prevailing wisdom in the database community was to work only with precise, valid data. Unfortunately,

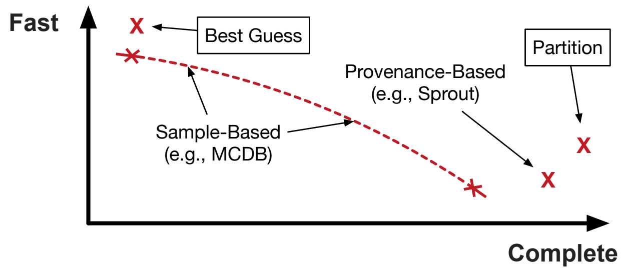


Fig. 1: **Overview of of evaluation strategies — Performance vs Utility. New contributions of this paper are boxed.**

hiding uncertain information (e.g., behind NULL-value semantics) dramatically decreases the utility of a data set. A value representing a guess, such as data extracted from a web page or an email, is information that can still be incorporated into a user's decision process. So-called probabilistic databases [1] represent a more sophisticated approach to managing uncertain data by treating data as a distribution over *possible* database instances. However, work on probabilistic databases to date has focused largely on query processing semantics, typically through algorithms for computing distributions of possible query results. In this work, we instead approach probabilistic databases by starting "in the last-mile," that is, by examining how users consume uncertain information. As a result of a user study, we find that simple, low-bandwidth signaling like text color is an effective and efficient way to communicate uncertainty in data. We also find that details like ranges and confidence bounds can make decisions harder for *some* users.

Armed with this information, we revisit existing strategies for evaluating answers to probabilistic database queries within a classical deterministic DBMS. To date, there have been two dominant strategies: provenance- and sample-based. Provenance-based evaluation as in Sprout [2] constructs a lossless representation of the distribution for a query's results. This representation can then be used to compute statistical metrics like row confidence to arbitrary precision. In short, provenance-based strategies can support a wide range of uncertainty representations. By contrast, sampling-based strategies like MCDB [3] can be faster (depending on number of samples), but do not produce a complete characterization of the result distribution and can not be as expressive as a provenance-based technique. This tradeoff is illustrated in Figure 1.

In short, provenance-based strategies can support a wider range of uncertainty representations, while sampling-based

[1]http://money.cnn.com/2016/04/11/pf/john-oliver-credit-reports/index.html
[2]http://www.nytimes.com/2010/01/14/nyregion/14watchlist.html

strategies are faster. The key insight of this paper is that for the simplest representations, which our user study shows can still be effective, both strategies are still computing too much. Thus, we propose a 2-pass approach to probabilistic query processing. In the first pass, we quickly generate a simple summary of query results to show to the user. Then, in the background, we begin constructing the full distribution as in a provenance based system. We propose evaluation strategies for each phase: (1) inline, a lightweight probabilistic evaluation strategy that is sufficient to enable lightweight representations and (2) partition, a more heavyweight strategy that can produce more detailed results. We implemented these strategies in the Mimir probabilistic database framework, and use the PDBench [4] probabilistic database benchmark to evaluate their performance.

A. *Data Uncertainty and Ambiguity*

Uncertainty, imprecision, incompleteness, or ambiguity in data arises in a number of settings. Consider the following examples.

Imprecise data sources. Data obtained through crowdsourcing [5], sensors, or privacy-preserving data releases is explicitly uncertain. Often the data source defines either bounds or a distribution for potential errors in the data.

Heuristic extraction. Numerous techniques have arisen for extracting structured data or relationships from prose, HTML `<table>` tags, JSON documents, and social networks. These techniques are typically heuristic — they are not guaranteed to produce correct results in all circumstances and may identify ambiguities in the source data that require human intervention.

Data cleaning. Data cleaning, for example through constraint enforcement [6] or entity de-duplication [7] typically creates multiple, ambiguous interpretations of source data that a heuristic must select from.

Approximate query processing. In interactive settings, it is often beneficial to trade result accuracy for improved latency [8].

Fuzzy queries Non-SQL query interfaces like natural language, gestures, or keyword search frequently admit ambiguous interpretations. A single query might have many possible result sets, each valid under some interpretation of the query.

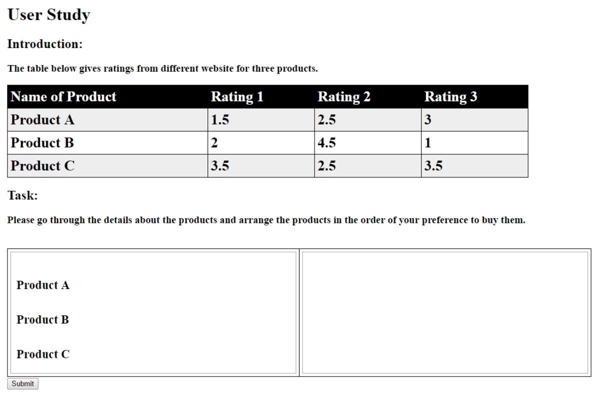
Uncertainty in relational data is typically categorized into one of three forms: (1) Uncertainty about the value of a given attribute in an existing row (attribute-level), (2) Uncertainty about whether a specific row should be present in the database (row-level), or (3) The full set of possible rows in the database may be unknown or infinite (open-world uncertainty). In this paper, we focus primarily on presenting forms of attribute-level uncertainty such as dependency violation repairs, domain violation repairs, extraction errors, or low-quality data sources.

B. *Overview*

The key insight driving our design is that simple representations of uncertainty are sufficient for some use cases. This insight is validated by the user study presented in Section II. In Section III, we define a taxonomy of uncertainty representations relative to what information is needed to generate each, and categorize existing probabilistic database systems with respect to which representations they can produce. Finally, in Section IV we leverage representations with lower information requirements to create a 2-pass evaluation strategy that simultaneously produces a quick simple representation and a slow detailed representation of uncertain query results. Concretely, the contributions of this paper are: (1) We describe the results of a user study assessing the effectiveness and efficiency of attribute-level uncertainty representations. (2) We outline a taxonomy of representations of uncertainty. (3) We describe a query evaluation strategy for user-focused probabilistic query processing. (4) We implement these strategies in the Mimir probabilistic database framework [9], [10] and demonstrate how the choice of representation affects query performance.

	Rating Source		
Product	**Buybeast**	**Amazeo**	**Targe**
Samesung	3.5 – 4.5	3.0	3.5±1
Magnetbox	2.5	2.5	3.0
Mapple	5.0$\star$	3.5	5.0

Fig. 2: **Example uncertainty representations.**



User Study

Introduction:

The table below gives ratings from different website for three products.

Name of Product	Rating 1	Rating 2	Rating 3
Product A	1.5	2.5	3
Product B	2	4.5	1
Product C	3.5	2.5	3.5

Task:

Please go through the details about the products and arrange the products in the order of your preference to buy them.

Product A

Product B

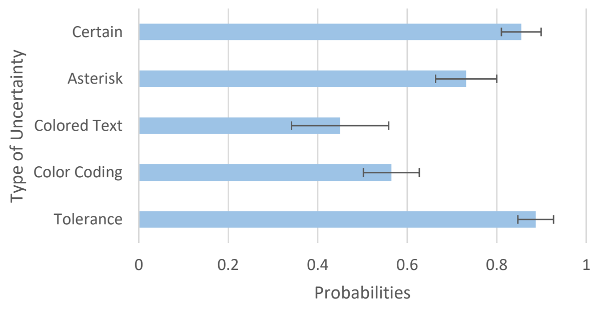
Product C

Fig. 3: **User Interface.**

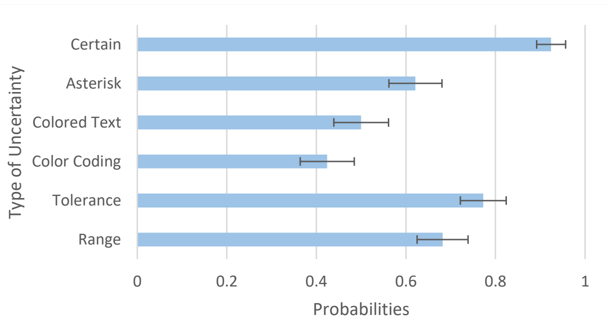
II. User Interface

Our first aim is to design a user interface for presenting query results with attribute-level uncertainty, optimizing for three objectives: (1) *Familiarity*: Replicate as much of the deterministic querying experience as reasonable, (2) *Effectiveness*: Get users to incorporate data uncertainty into decisions they make based on uncertain query results, and (3) *Efficiency*: Minimize the cognitive burden of interpreting uncertainty in results. In the interest of optimizing for familiarity, we begin with a classical deterministic interface: a table of query results. We focus our efforts on modifying individual table cells to communicate uncertainty. We select a representative sample of representations, each illustrated in Figure 2.

1) **Asterisk**: Uncertain values were marked with an asterisk (e.g., Mapple's Buybeast rating).
2) **Colored text**: The text of uncertain values was colored red (e.g., Magnetbox's Amazeo rating).
3) **Color coding**: The cells containing uncertain ratings were given a red background (e.g., Magnetbox's Targe rating).



(a) Preliminary Study Results (from [11])



(b) Results from This Study

Fig. 4: **Agreement with BestOf3 order**

4) **Tolerance**: Uncertain values were presented with an error bound (e.g., Samesung's Targe rating).
5) **Range**: Uncertain values were presented as a range of values (e.g., Samesung's Buybeast rating).

The first three representations are simple low-bandwidth encodings. Each communicates precisely 1 bit of information about each attribute, albeit with varying levels of subtlety. Coloration, and in particular red is a common warning signal — Coloring text is subtle, while coloring an entire cell is more observable while decreasing the contrast of the value and making it harder to see. Symbols like asterisks are frequently used in tabular data for footnotes. The remaining two representations carry a higher level of detail: a range of possible values, expressed as a median and range in one case and as upper and lower bounds in the other.

A. *Study Design*

The two primary questions that we sought to answer for each of the representations of uncertainty were (1) Is the representation *effective* at communicating uncertainty, and (2) What is the *cognitive burden* of interpreting the representation? To address these questions we conducted a series of user studies in which participants were presented with a web form that contained a 3x3 grid showing three ratings each for three products. The participants were told that the ratings came from three different sources and were normalized to a scale of 1 to 5, with 5 being best and 1 being worst. Given this information, participants were asked to evaluate the products for purchase by ranking the products in the order of their preference. The interface used in the study is illustrated in Figure 3.

Interactions with the web-form — such as product selection, re-ordering the product list, and submitting the participant's final order — were logged along with time stamps. In addition to interactions with the web form, the experiment also used a think-aloud protocol. The think-aloud protocol is defined to be a process in which participants are asked to verbalize their thought processes while performing the task. Audio logs were then transcribed and the anonymized transcriptions tagged and coded for analysis.

Control Trials. Ratings for each product were generated randomly for each trial, but biased to elicit a specific, predictable ranking order. We call one product *roughly better* than a second if it has (1) one extremely favorable rating (at least one point higher), (2) one slightly favorable rating (0, 0.5, or 1 point higher), and (3) one slightly unfavorable rating (0, 0.5, or 1 point lower). We call a product rating grid *valid* if there exists a sequence of products where each product is *roughly better* than the next in the sequence. For each trial, we sampled rating grids from a uniform random distribution on the range $[0, 5]$ with increments of 0.5, and used rejection sampling to ensure only *valid* grids. As shown in Figures 4a and 4b, this sampling process elicited the expected ordering from participants roughly 90% of the time. From this point we will refer to this elicited ordering as the best two-out-of-three ratings, or **BestOf3** ordering.

Experimental Trials. Each experimental trial evaluated one of the five forms of uncertainty described above. In each experimental trial, the grid was generated exactly as in the control trial. However, between 2 and 4 randomly chosen values were *labeled* as being uncertain. Participants were also informed that these fields were uncertain. For the asterisk, colored text, and color coding representations, the label was simply applied to the affected cells. For the range and tolerance representations, the presented interval was uniformly selected from $\{\pm 0.5, \pm 1, \pm 1.5\}$. To emphasize, the sole difference between an experimental trial and a control trial was the addition of labels marking some ratings as uncertain and not how the displayed (or median) ratings were generated.

Scale. A total of 22 participants drawn from the entire student body of the University at Buffalo participated. Participants were asked to complete three rounds of survey, with each round consisting of six trials: One control trial and one experimental trial for each of the five representations. Thus, we collected 66 rankings per representation (and for the control) for a total of 198 pairwise product comparisons each.

For comparison, we also review some results from a preliminary study [11]. The preliminary study included 14 participants drawn exclusively from the University at Buffalo's Department of Computer Science and Engineering and did not evaluate the Range representation.

B. *Quantitative Assessment*

As mentioned in Section II-A, a participant's interactions with the web-form were logged, including the participant's final ranking and the time taken to reach it. We first evaluate

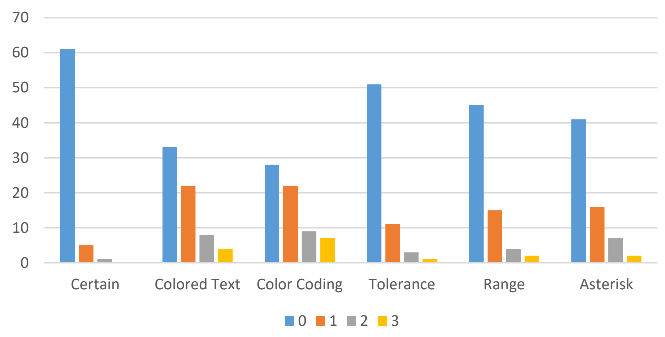


Fig. 5: **Pairwise Deviation from BestOf3**

the representations quantitatively for (1) effectiveness, by comparing participant rankings with the BestOf3 raking and (2) efficiency, by measuring the time taken to complete the trial.

Effectiveness. A product ranking that is closer to random relative to BestOf3 is expected if a representation of uncertainty is effective. Figure 4 summarizes our results, showing the probability of total agreement between the participant-selected ordering and the BestOf3 ordering. Standard deviations are computed under the assumption that agreement with BestOf3 follows a Beta-Bernoulli distribution. A 16.7% agreement would be consistent with a purely random chance of total agreement with BestOf3. The deterministic baseline (**Certain**) shows a consistent, roughly 92% agreement with BestOf3. Figure 5 illustrates the same data as Figure 4b, but considers the number of *pairwise* disagreements in relative ranking rather than complete agreement. Both colored text and color coding significantly altered participant behavior (50% and 42% agreement with BestOf3). This is comparable to our preliminary study where colored text and color coding where 45% and 56% in agreement with BestOf3 respectively. There is an increase in agreement with BestOf3 for colored text, but remains within one standard deviation of the preliminary study. In the new study, color coding agrees significantly less with BestOf3; This may be attributable to the difference in study populations. In both studies color coding was very effective at altering user behavior. The range representation was not as effective at altering participant behavior (68% agreement), although we would expect a closer agreement, since averaging the range value would cause the participant to follow BestOf3. As discussed in our qualitative analysis below, this may be attributable to participants ignoring the uncertain values altogether or selecting the lowest or highest value. The preliminary study reveals the tolerance representation showed a consistent (89%) agreement with BestOf3 whereas we can see a decrease in the agreement (77%) in the broader study. Our analysis shows that this might be because the Non CS students took significant time in interpreting tolerance values. Asterisks were not as effective at altering participant behavior (62% agreement), which confirms with results from the preliminary study [11] with asterisk at 73% agreement.

Efficiency. We measure time taken for each form of uncertainty as a proxy for cognitive burden. Figure 6 illustrates time taken by users to complete each individual ranking task. In the case of CS participants, time taken per representation was relatively consistent across all forms of uncertainty except tolerance. The slowest trial for both CS Figure 6a and Non-CS Figure 6b participants was tolerance. As seen in Figure 6b, participants from a Non-CS background took longer to infer tolerance, certain and range representations. However, non-CS participants also displayed a quicker decision compared to CS participants in case of asterisk, colored Text and color coding representations. The comparison might suggest that being familiar with the representation (tolerance and ranges) reduces the cognitive burden of interpreting uncertainty.

C. *Qualitative Assessment*

We next analyze the results of the think-aloud protocol used in the study. In general, consistency in the rating sources and the products was considered as secondary sources of feedback regarding data quality. For example, if Source 1 had uncertain ratings for two products, then participants were more likely to discard it as uninformative and base their rating solely on the other two sources. If the ratings for a product had wide (4.5, 2, 1) range then the product was considered unreliable by a few participants. Participants were encouraged to state whether they were using BestOf3 or taking an average of the three ratings in order to rank the products.

For a principled analysis of the think-aloud results, we transcribed and examined the collected audio logs. Utterances were annotated with tags, broadly grouped into three categories: (1) **Risk** tags annotate utterances where the participant describes an optimistic or pessimistic decision strategy that incorporates uncertain values. (2) **Comfort** tags annotate utterances where the participant indicates a positive or negative emotion, either towards the deterministic or non-deterministic data. (3) **Context** tags annotate utterances where the participant attempts to use context (the value domain, other ratings, etc. . .) to infer a cause for or interpretation of the uncertain values.

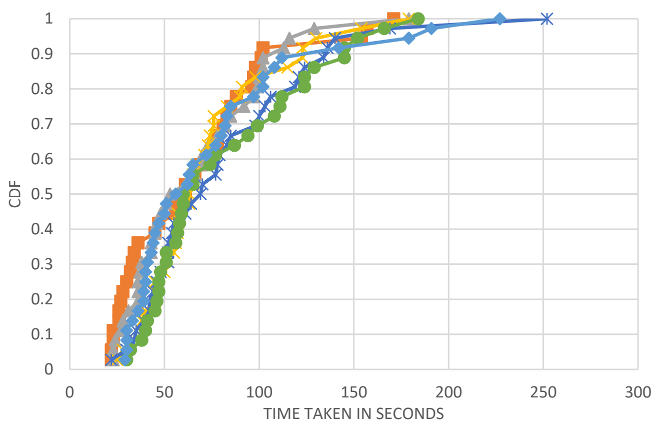
Risk-Aversion vs. Risk-Favor. Figure 7 illustrates the effect of different presentations of uncertainty on participants' willingness to take risks and choose a product that has uncertain ratings.

1) `Risk-Aversion`: The participant indicates that they are making a decision that avoids risk or indicates a reduced preference for an option based on too much uncertainty.
2) `Risk-Favor`: The participant indicates that they are making a decision that takes on risk or they believe that the risk is minimal, although their decision involves uncertainty.

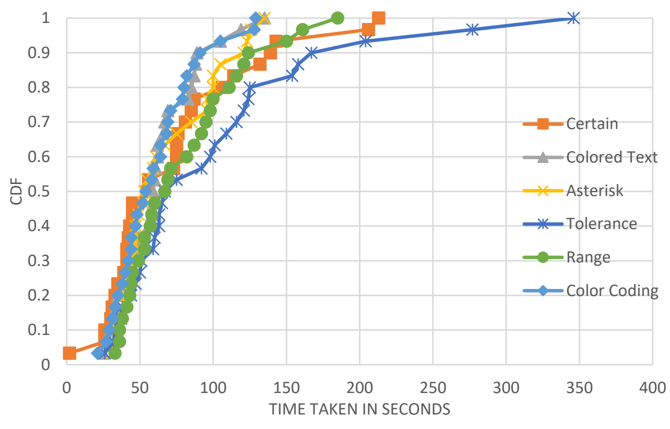
Participants were more likely to indicate risk-taking behavior with color coding, and less likely to indicate risk-taking behavior with the tolerance or range representations.

Comfort vs. Discomfort. Figure 8 illustrates positive and negative emotional reactions expressed by participants with regards to the data and the type of uncertainty.

1) `Discomfort-Data`: The participant expresses discomfort with ambiguity in the (raw, certain) data or the difficulty of the decision process.
2) `Comfort-Data`: The participant expresses comfort with the data being presented and/or being at-ease with the decision they are being asked to make.



(a) CS participants



(b) Non CS participants

Fig. 6: **Time taken per form of uncertainty. Graphs show cumulative distributions for cs vs. non-cs participants.**

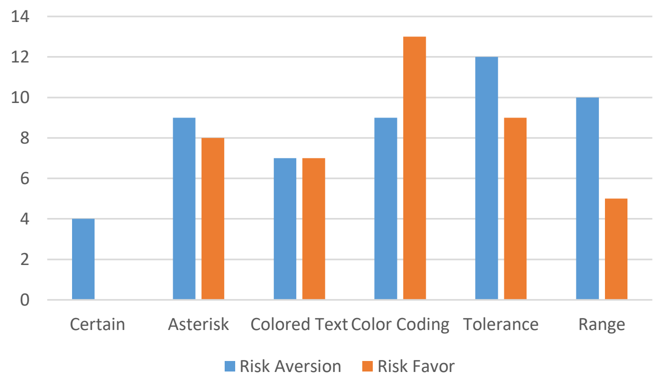


Fig. 7: **Risk Aversion vs. Risk Favor**

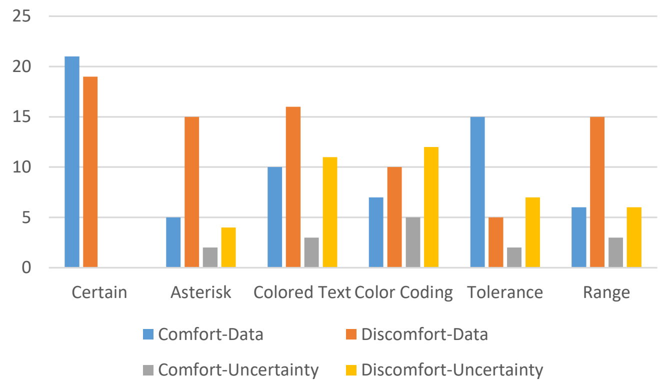


Fig. 8: **Comfort vs. Discomfort**

3) `Discomfort-Uncertainty`: The participant expresses discomfort or a dislike of the uncertainty in the data.
4) `Comfort-Uncertainty`: The participant expresses comfort with or a like of the uncertainty in the data.

The color coding representation made the participants most uncomfortable followed by colored text representation as seen in Figure 8 whereas Tolerance was considered as the most comfortable representation of uncertainty. Asterisk, Colored text and Ranges induced the same level of discomfort with data. Users expressed the most discomfort with color coding representations, followed by colored text as seen in Figure 8. Tolerance was considered the most comfortable representation of uncertainty. Asterisk and ranges induced the same level of discomfort with the data.

Context-X vs. Uncertainty-X. Uncertain values were ignored on a larger scale in color coding compared to other representations, followed by asterisk and colored text as seen in Figure 9. Uncertainty was also considered irrelevant in these three cases.

1) `Context-Row`: The participant indicates a change in behavior (e.g., discarding or retaining) with respect to an uncertain value based on other ratings from the same source for the same product.
2) `Context-Domain`: The participant expresses a desire to select a product because its uncertain ratings are low (and can only increase), or a desire to avoid a product because its uncertain ratings are high (and can only decrease).
3) `Uncertainty-Callout`: The participant vocalizes a form of uncertainty perceived in the data.
4) `Uncertainty-Ignored`: The participant expresses an intent to ignore, discard, exclude, or otherwise disregard values that are uncertain. The participant may also explicitly state that a decision is based exclusively on certain data.
5) `Uncertainty-Irrelevant`: The participant expresses awareness that the uncertainty in the values does not affect their decision.

III. Probabilistic Query Evaluation

We next discuss the state of the art in in-situ probabilistic query evaluation strategies and contrast it with the needs of the query result presentations examined in Section II. Concretely, we aim to enable probabilistic query processing within a classical, deterministic database through query rewriting. The five representations in the user study fall into two categories: **1-bit** representations (Asterisk, Color Coding and Colored

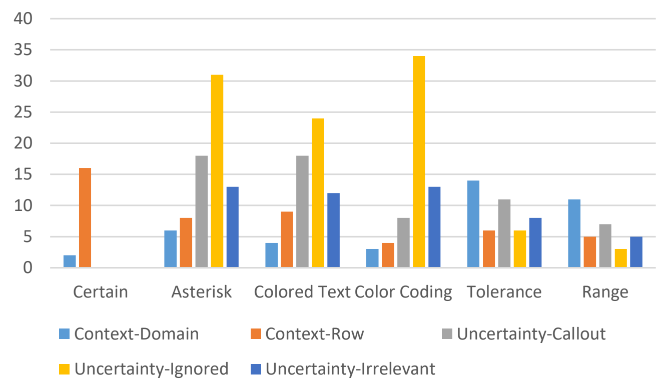


Fig. 9: **Context vs. Uncertainty**

Text) that identify only which values are uncertain, and **range** representations (Tolerance and Range) that provide a range of values. More **complex** representations than addressed in our user study (e.g., communicating inter-attribute correlations) are also possible.

A. *Representation-Oriented Query Processing*

These categories define a hierarchy of evaluation strategies: **1-bit** evaluation strategies should emit a single query result annotated with which values are uncertain, **range** evaluation strategies should emit a (probabilistic) range of possible values, and **complex** evaluation strategies should be able to produce a lossless encoding of the full distribution of query results.

Generating a *complex* representation. This class of representations requires computing a full, lossless representation of the result distribution. Although this is the most general class of evaluation semantics, lossless representations are frequently reduced to simpler summary representations before presentation to the user. For example, in both PIP [12] and Sprout [13], users request specific summaries like probabilistic aggregates, expectations, or confidence values through the query. Other examples of systems that support complex representation semantics include MYSTIQ [14], Trio [15], MayBMS [16], and Orion 2.0 [17], as well as the **partition** query evaluation strategy outlined in Section IV.

Generating a *range* representation. This class of representations requires computing a representative range of values. The range presented may be either exact or probabilistic (e.g., ϵ-δ bounds). Especially for data in a continuous domain, probabilistic ranges are frequently more useful as many common continuous value distributions have no hard upper or lower bounds. Support complex representations implies support for both exact and probabilistic bounds. As we discuss in greater detail below, the MCDB system [3] also supports probabilistic range representations.

Generating a *1-bit* representation. This class of representations requires computing a single, representative value for each cell in the result and a boolean value for each cell that determines whether the value differs across possible worlds. A key question in generating 1-bit representations is the choice of representative value, most logically the mode or mean of some distribution. We note two reasonable choices: The result value that maximizes likelihood in the prior, pre-query distribution, or the result value that maximizes likelihood in the posterior, query result distribution.

Example 1: Consider two independent boolean variables $P(A) = P(B) = 0.4$. The most likely possible world is the one where A and B are both false, and the value of $A \vee B$ in this "best guess" world is false. Conversely $P(A \vee B) = 0.64$. Although the result of this query is most likely true, there is a possibility that changing the best guess based on query may be confusing to users[3].

Support for complex representations implies support for both prior and posterior distributions. MCDB's tuple bundles are also capable of supporting a 1-bit representation based on the posterior distribution, and a probabilistic correctness guarantee on the bit. Finally, the **inline** query evaluation strategy proposed in Section IV supports 1-bit representations based on the prior distribution.

B. *Existing Techniques for Probabilistic Query Processing*

Before discussing the current state of the art, we first need to outline some background and terminology. Query semantics over a non-deterministic database are given by *possible worlds semantics* as an extension of deterministic query semantics. A deterministic query Q applied to an uncertain database defines a set of possible results $Q(\mathcal{D}) = \{\ Q(D) \mid D \in \mathcal{D}\ \}$. Note that these semantics are agnostic to the data representation, query language, and number of possible worlds $|\mathcal{D}|$. The possible worlds of an uncertain database may be annotated with a probability measure $P : \mathcal{D} \rightarrow [0, 1]$ to form a *probabilistic database* $\langle\ \mathcal{D}, P\ \rangle$. The probability measure also induces a distribution over possible result relations R for the query:

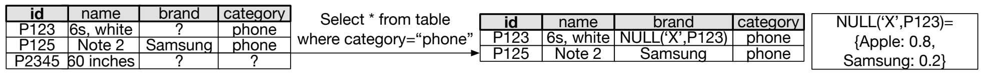
$$P[Q(\mathcal{D}) = R] = \sum_{D \in \mathcal{D}\ :\ Q(D) = R} p(D)$$

A single uncertain database $\mathcal{D}$ may be characterized as a (potentially infinite) set of possible deterministic database instances $D \in \mathcal{D}$, also known as *possible worlds*. It is common to assume that possible worlds share a single schema $sch(\mathcal{D})$.

There are several existing solutions on query processing on uncertain data. MCDB [3] allows the user to define the distribution of the uncertain data and samples from this distribution to represent uncertain data in "tuple bundles", or fixed-size arrays of tuples, all having the same schema. Each element of the bundle represents the tuple in one (sampled) possible world. Tuple bundles allow MCDB to evaluate queries over all sampled worlds in parallel to produce samples from the distribution of possible results, and can be implemented in any classical DBMS that supports UDFs [12].

C-Tables. The number of possible worlds necessary to describe a data set may be very large (or in some cases infinite). Fortunately, each possible world can often be interpreted as a collection of (mostly) orthogonal decisions. For many sources of uncertainty, these decisions follow one of two patterns. The first examines the existence of a specific row in a given possible world. The second evaluates the value of an

[3]We leave to future work the question of whether prior or posterior distributions are more intuitive to database users.

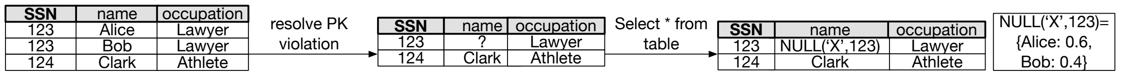


id	name	brand	category
P123	6s, white	?	phone
P125	Note 2	Samsung	phone
P2345	60 inches	?	?

id	name	brand	category
P123	6s, white	NULL('X',P123)	phone
P125	Note 2	Samsung	phone

NULL('X',P123)=
{Apple: 0.8,
Samsung: 0.2}

(a) Some source data is missing.



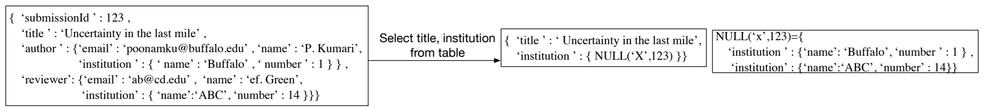
SSN	name	occupation
123	Alice	Lawyer
123	Bob	Lawyer
124	Clark	Athlete

SSN	name	occupation
123	?	Lawyer
124	Clark	Athlete

SSN	name	occupation
123	NULL('X',123)	Lawyer
124	Clark	Athlete

NULL('X',123)=
{Alice: 0.6,
Bob: 0.4}

(b) Data is from an unreliable source.



```
{ 'submissionId ' : 123 ,
  'title ' : 'Uncertainty in the last mile' ,
  'author ' : {'email' : 'poonamku@buffalo.edu' , 'name' : 'P. Kumari',
               'institution ' : { ' name' : 'Buffalo' , ' number ' : 1 } } ,
  'reviewer': {'email' : 'ab@cd.edu' , 'name' : 'ef. Green',
               'institution' : { 'name':'ABC', 'number' : 14 }}}
```

```
{ 'title ' : ' Uncertainty in the last mile',
  'institution ' : { NULL('X',123) }}
```

```
NULL('x',123)={
  'institution ' : {'name': 'Buffalo', 'number ' : 1 } ,
  'institution' : {'name':'ABC', 'number' : 14}}
```

(c) A fuzzy query on certain data.

Fig. 10: **Input data and labeled null representation, Q and expecting query result** $\{Q(D), P\}$**.**

attribute associated with a specific row in a the possible world. These two patterns, respectively termed row- and attribute-level uncertainty, form the basis of a factorized encoding called C-Tables. A C-Table [18] is a relation instance where each tuple is annotated with a formula ϕ, a propositional formula over an alphabet of variable symbols σ. ϕ is often called a local condition and the symbols in σ are referred to as labeled nulls, or just variables. Intuitively, for each assignment to the variables in Σ we obtain a possible relation containing all the tuples whose formula ϕ is satisfied. For example:

	Product				
	pid	**name**	**brand**	**category**	ϕ
t_1	P123	Apple 6s	Apple	phone	$x_1 = 1$
t_2	P123	Apple 6s	Cupertino	phone	$x_1 = 2$
t_3	P125	Note2	Samsung	phone	$\top$

$$x_1 = \begin{cases} 1 : 0.3 \\ 2 : 0.7 \end{cases}$$

The above C-Table defines a set of two possible worlds, $\{t_1, t_3\}$, $\{t_2, t_3\}$, i.e. one world for each possible assignment to the variables in the one-symbol alphabet $\Sigma = \{x_1\}$. Notice that no possible world can have both t_1 and t_2 at the same time. Finally, a C-Table may be combined with a probability measure to obtain a Probabilistic C-Table, or PC-Table [19].

VG-Relational Algebra. VG-RA (variable-generating relational algebra) [12] is a generalization of positive bag-relation algebra. Primitive-value expressions in VG-RA (i.e., projection expressions and selection predicates) can contain a new algebraic operator $Var(\ldots)$, which dynamically introduce new unique skolem symbols in Σ, deterministically derived from the function's parameters, which serve as labeled nulls. These skolem symbols are not useful on their own; VG-RA expressions are accompanied by a *model* that associates new skolem symbols with probability distributions. Hence, VG-RA can be used to define new PC-Tables.

Mimir. In prior work, we proposed the Mimir system [10], which uses VG-RA queries to declare probabilistic constraint repairs as views. Probability measures are constructed using off-the-shelf machine learning tools and techniques.

Example 2: Consider the table in Figure 10a, which is missing several values for the brand and category attributes. Mimir defines a probabilistic repair for these missing values as the VG-RA query:

$$\pi_{id \leftarrow id,\ name \leftarrow name,\ brand \leftarrow f(brand),\ cat \leftarrow f(cat)}(Product)$$

Here, f denotes a validate and repair expression:

$$f(x) \equiv \textbf{if } x \texttt{ is null } \textbf{then } Var(x, \texttt{ID}) \textbf{ else } x$$

If the value is present, is is left unchanged. If it is null, f replaces it with a labeled null created by the skolem function $Var(x, \texttt{ROWID})$ — One unique variable is created for every column and row. The value of this labeled null is given by a model independently trained on the same data[4].

The repaired tables becomes a view and can be queried directly. A key feature of VG-RA is that any query may be normalized into the form: $\pi_A(\sigma_\phi(Q(D)))$, where $Q(D)$ is a classical, completely deterministic query and Var operators appear only in ϕ and A. This is not only true for the repair views, but for queries over them as well. Thus, to evaluate a query over a repair view, the query is first normalized. The deterministic part $Q(D)$ is evaluated by a classical RDBMS, while π_A and σ_ϕ are evaluated by a thin middleware shim between the user and the database. The shim query, $\mathcal{F}_{A,\phi}(R) \equiv \pi_A(\sigma_\phi(R))$, the joint distribution over its skolem symbols, and the results of $Q(D)$ together form a complete, lossless description of the distribution of possible results. Hence Mimir's default evaluation strategy is sufficient for **complex** representations.

IV. Towards Interactive Probabilistic Queries

Supporting interactive data exploration requires rapid query responses. As shown in Section II, even simple representations of uncertainty can be sufficient for users to make a reasonable choice based on the data that is available. Conversely, the user may actually need more detailed information. In this section we outline the design of a 2-pass query evaluation strategy for probabilistic databases. First, a simplified form of the full probabilistic query is run to generate a simple 1-bit representation for immediate display to the user. Simultaneously in the background, the system generates a full lossless encoding of the result distribution, which can be used to quickly construct supplemental details. Specifically, we outline two new evaluation strategies, one for each pass: (1) The **inline** evaluation strategy quickly produces the system's best guess for what the result should be and (2) The **partition** evaluation strategy efficiently constructs a complete representation of the result distribution.

[4]Mimir uses the popular Weka open source library for this purpose.

A. Pass 1 Strategy — Inline

During best-guess query evaluation, each labeled null is replaced by a single, deterministic best-guess. In the simplest case, the best-guess selection heuristic (e.g., a lookup on a pre-trained classifier) can be embedded into the database as a user-defined function (UDF). When this is possible, the query can be evaluated almost unchanged by a classical database engine by replacing all Var terms with their matching UDF. However, since not all DBMSes currently support programming languages in-database and an alternative strategy may be needed.

A second concern is that simply computing the best guess result not sufficient to construct a 1-bit representation of the uncertainty in that result. To identify which cells should be marked, we annotate the result with an additional set of columns marking which values are deterministic or not.

The third and final concern is linking results produced in the best-guess analysis phase to results produced by the second-pass evaluation run. This is accomplished by extending the query with additional provenance attributes.

Inlining Best-Guess Values. If possible, best-guess estimates can be inlined into the query through UDFs. However, if the selection heuristic can not be embedded into a UDF or if the database does not support them, a more general solution is to materialize a table of best-guesses.

To populate the best guess tables, we simulate the execution of the query that generates the best guesses, identifying every variable instance used and materializing into one table for each skolem function. When a non-deterministic query is run, all references to Var terms are replaced by nested lookup queries which read the values for Var terms from the corresponding best guess tables. As a further optimization, the in-lined lens query can also be pre-computed as a materialized view.

Example 3: Recall that in Example 2 a probabilistic repair query defines a view where missing values are replaced by labeled nulls. For instance, the missing brand of product $P123$ will instantiate a null $NULL_{brand}['P123']$. As a pre-processing step when this view is first created, we load the best guess values for these nulls into a lookup table `BrandGuess(param1,value)` Mentions of variables in queries over the lens are replaced with lookups on the guess table. A SELECT query for Product.brand would either be inlined as

```
SELECT CASE WHEN p.brand IS NULL
       THEN BrandGuess(ID)
       ELSE p.brand END AS BRAND FROM Product
```

or, if UDFs are not available, it is rewritten as

```
SELECT CASE WHEN p.brand IS NULL THEN bg.data
       ELSE p.brand END AS BRAND
FROM Product p LEFT OUTER JOIN
     BrandGuess bg ON bg.param1 = p.ID,
```

1) Result Determinism: The 1-bit class of representations requires that we compute, for each cell of the output, whether it is certain or not. As an additional rewrite, we also compute the 1-bit determinism of the row as a whole. We next describe the process of annotating a query to track this information.

Concretely, before best guess values are inlined into a query Q with schema $sch(Q) = \{a_i\}$, we rewrite it into a new query $[[Q]]_{det}$ with schema $\{a_i, D_i, \phi\}$. Each D_i is a boolean-valued attribute that is true for rows where the corresponding a_i is deterministic. ϕ is a boolean-valued attribute that is true for rows deterministically placed in the result set. We refer to these two added sets of columns as attribute- and row-determinism metadata, respectively. Query $[[Q]]_{det}$ is derived from the input query Q by applying the operator specific rewrite rules described below, in a top-down fashion starting from the root operator of query Q.

Projection. The projection rewrite relies on Algorithm 1, which rewrites columns according to the determinism of the input. The algorithm works by recurring through an arithmetic expression and detecting dependencies on Var terms. A key feature of the algorithm is the observation that non-linear algebraic operators ($\vee$, $\wedge$ and `CASE` expressions) can remove dependencies on non-deterministic values. For example, if the left-hand side of a boolean and is deterministically false, the right-hand side is irrelevant. Algorithm 1 identifies such cases and constructs a boolean formula that is true if the algorithm is deterministic and false otherwise. Note also that columns in the expression are replaced by a dependency on the recursively computed determinism of the child operator.

The rewritten projection is computed by extending the projection's output with determinism metadata. Attribute determinism metadata is computed using the expression returned by `isDet` and row determinism metadata is passed-through unchanged from the input.

$$[[\pi_{a_i \leftarrow e_i}(Q)]]_{det} \mapsto \pi_{a_i \leftarrow e_i, D_i \leftarrow \texttt{isDet}(e_i), \phi \leftarrow \phi}([[Q]]_{det})$$

Algorithm 1 `isDet`(E)

```
In: $E$: An arithmetic expression that may contain $Var$ terms.
Out: An expression that is true when $E$ is deterministic.
  if $E \in \{\mathbb{R}, \top, \bot\}$ then
    return $\top$
  else if $E$ is $Var$ then
    return $\bot$
  else if $E$ is $Column_i$ then
    return $D_i$
  else if $E$ is $\neg E_1$ then
    return isDet($E_1$)
  else if $E$ is $E_1 \vee E_2$ then
    return $(E_1 \wedge \texttt{isDet}(E_1)) \vee (E_2 \wedge \texttt{isDet}(E_2))$
              $\vee\ (\texttt{isDet}(E_1) \wedge \texttt{isDet}(E_2))$
  else if $E$ is $E_1 \wedge E_2$ then
    return $(\neg E_1 \wedge \texttt{isDet}(E_1)) \vee (\neg E_2 \wedge \texttt{isDet}(E_2))$
              $\vee\ (\texttt{isDet}(E_1) \wedge \texttt{isDet}(E_2))$
  else if $E$ is $E_1\ \{+, -, \times, \div, =, \neq, >, \geq, <, \leq\}\ E_2$ then
    return $(\texttt{isDet}(E_1) \wedge \texttt{isDet}(E_2))$
  else if $E$ is if $E_1$ then $E_2$ else $E_3$ then
    return $\texttt{isDet}(E_1) \wedge ($   $(E_1 \wedge \texttt{isDet}(E_2))$
                       $\vee(\neg E_1 \wedge \texttt{isDet}(E_3))$  $)$
```

Selection. Like projection, the selection rewrite makes use of `isDet`. The selection is extended with a projection operator that updates the row determinism metadata if necessary.

$$[[\sigma_\psi(Q)]]_{det} \mapsto \pi_{a_i \leftarrow a_i, D_i \leftarrow D_i, \phi \leftarrow \phi \wedge \texttt{isDet}(\psi)}(\sigma_\psi([[Q]]_{det}))$$

Cross Product. Result rows in a cross product are deterministic if and only if both of their input rows are deterministic. Cross products are wrapped in a projection operator that combines the row determinism metadata of both inputs, while leaving the remaining attributes and attribute determinism metadata intact.

$$[[Q_1 \times Q_2]]_{det} \mapsto \pi_{a_i \leftarrow a_i, D_i \leftarrow D_i, \phi \leftarrow \phi_1 \wedge \phi_2}([[Q_1]]_{det} \times [[Q_2]]_{det})$$

Union. Bag union already preserves the determinism metadata correctly and does not need to be rewritten.

$$[[Q_1 \cup Q_2]]_{det} \mapsto [[Q_1]]_{det} \cup [[Q_2]]_{det}$$

Relations. annotate each attribute and row as being deterministic. During the base case of the rewrite, we annotate each attribute and row as deterministic once we arrive at a deterministic relation.

$$[[R]]_{det} \mapsto \pi_{a_i \leftarrow a_i, D_i \leftarrow \top, \phi \leftarrow \top}(R)$$

Optimizations. These rewrites are quite conservative in materializing the full set of determinism metadata attributes at every stage of the query. It is not necessary to materialize every D_i and ϕ if they can be computed statically based solely on each operator's output. For example, consider a given D_i that is data-independent, as in a deterministic relation or an attribute defined by a Var term. D_i has the same value for every row, and can be factored out of the query. A similar property holds for Joins and Selections, allowing the projection enclosing the rewritten operator to be avoided.

2) Linking the Phases: The queries evaluated by the backend database for each of the two evaluation passes are different. When additional information is required, it may be necessary to link individual rows of the inlined result with the corresponding row of the phase 2 result. This is accomplished by adding a simple provenance marker to the query.

As the basis for provenance markers, we use an implicit, unique per-row identifier attribute called ROWID supported by many popular database engines. When joining two relations in the in-lined query, their ROWIDs are concatenated (we denote string concatenation as $\circ$):

$$Q_1 \times Q_2 \mapsto \pi_{a_i \leftarrow a_i, \texttt{ROWID} \leftarrow \texttt{'('} \circ \texttt{ROWID}_1 \circ \texttt{')('} \circ \texttt{ROWID}_2 \circ \texttt{')'}}(Q_1 \times Q_2)$$

When computing a bag union, each source relation's ROWID is tagged with a marker that indicates which side of the union it came from:

$$Q_1 \cup Q_2 \mapsto \pi_{a_i \leftarrow a_i, \texttt{ROWID} \leftarrow \texttt{ROWID} \circ \texttt{'+1'}}(Q_1) \cup \pi_{a_j \leftarrow a_j, \texttt{ROWID} \leftarrow \texttt{ROWID} \circ \texttt{'+2'}}(Q_2)$$

Selections are left unchanged, and projections are rewritten to pass the ROWID attribute through.

In addition to linking rows of the two result sets, provenance markers have another benefit. Selections for specific result rows can be pushed down into the pass-2 query, making it possible to obtained detailed distribution information for just one row of the output. This process of unwrapping the marker, summarized in Algorithm 2, illustrates how a symmetric descent through the deterministic component of a normal form query and a provenance marker can be used to produce a single-row of Q'. The descent unwraps the provenance marker, recovering the single row from each join leaf used to compute the corresponding row of Q'.

Algorithm 2 unwrap(Q', id)

```
In: $Q'$: The deterministic component of a VG-RA normal form query.
In: $id$: A ROWID from the inlined query $Q$ that was normalized into $\mathcal{F}(Q'(D))$.
Out: A query to compute row $id$ of $Q'$
  if $Q'$ is $\pi(Q_1)$ then
    return $\pi(\texttt{unwrap}(Q_1, id))$
  else if $Q'$ is $\sigma(Q_1)$ then
    return $\sigma(\texttt{unwrap}(Q_1, id))$
  else if $Q'$ is $Q_1 \times Q_2$ and $id$ is $\texttt{(}id_1\texttt{)(}id_2\texttt{)}$ then
    return $\texttt{unwrap}(Q_1, id_1) \times \texttt{unwrap}(Q_2, id_2)$
  else if $Q'$ is $Q_1 \cup Q_2$ and $id$ is $id_1\texttt{+1}$ then
    return $\texttt{unwrap}(Q_1, id_1)$
  else if $Q'$ is $Q_1 \cup Q_2$ and $id$ is $id_1\texttt{+2}$ then
    return $\texttt{unwrap}(Q_2, id_1)$
  else if $Q'$ is $R$ then
    return $\sigma_{\texttt{ROWID}=id}(R)$
```

B. Pass 2 Strategy — Partition

Most existing provenance-based evaluation strategies [13]–[17] limit themselves to supporting finite, discrete data distributions. This is a necessary concession to efficiency, as non-deterministic joins over data drawn from a continuous distribution effectively devolve to cross products. However, since time is not as pressing a concern for the background task, our second pass can lift this restriction.

To avoid completely devolving to cross-product performance, our second-pass query evaluation strategy is based on the assumption that a comparatively small fraction of the user's data is uncertain. In this case, the backend database can be better utilized if the query's inputs are partitioned into deterministic and non-deterministic segments, each computed each independently, and unioned together at the end. For the deterministic partition of the data, joins can be evaluated as usual and other selection predicates can be satisfied using indexes over the base data. Non-deterministic partitions can be selected so that tuples in the partition share a common set of uncertain dependencies, simplifying evaluation

In this section, in the context of a specific subquery, we will use ϕ_i to represent a boolean expression that captures the lineage of attributes and rows in the subquery. We use ψ_i to denote the where clause for the sub-query. To partition a query $(Q(D))$, we begin with a set of partitions, each defined by a boolean formula ψ_i over attributes in $sch(Q)$. For each partition ψ_i we can simplify the selection condition ϕ of a query Q into a reduced form ϕ_i. We use $\phi[\psi_i]$ to denote the result of propagating the implications of ψ_i on ϕ. For example, (**if** X `is null` **then** $Var('X', ROWID)$ **else** $X)[X$ `is null`$] \equiv Var('X')$. For a set of partitions to be used to split a query into fragments it must be complete ($\bigvee \psi_i \equiv T$) and disjoint ($\forall i \neq j \;.\; \phi[\psi_i] \rightarrow \neg\phi[\psi_j]$).

Given a set of partitions $\Psi = \{\psi_1, \dots, \psi_N\}$, the partition rewrite transforms the original query into an equivalent set of partitioned queries as follows:

$$\begin{aligned}&(\mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi)(Q(D)))\\ &\quad\mapsto \mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi_{var,1})(\sigma_{\psi_1 \wedge \phi_{det,1}}(Q(D)))\\ &\quad\cup \dots \cup \mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi_{var,N})(\sigma_{\psi_N \wedge \phi_{det,N}}(Q(D)))\end{aligned}$$

where $\phi_{var,i}$ and $\phi_{det,i}$ are respectively the non-deterministic and deterministic conditions of ϕ (i.e., $\phi = \phi_{var,i} \wedge \phi_{det,i}$) for each partition. Partitioning then, consists of two stages: (1) Obtaining a set of potential partitions Ψ from the original condition ϕ, and (2) Segmenting ϕ into a deterministic filtering predicate and a non-deterministic lineage component.

Algorithm 3 naivePartition(ϕ)

```
In: $\phi$: A non-deterministic boolean expression
Out: $\Psi$: A set of partition conditions $\{\psi_i\}$
  $conditions \leftarrow \emptyset$
  $\Psi \leftarrow \emptyset$
  for (if $condition$ then $\alpha$ else $\beta$) $\in$ subexps($\phi$) do
    /* Check ifs in $\phi$ for candidate partition conditions */
    if isDet($condition$) $\wedge$ (isDet($\alpha$) $\neq$ isDet($\beta$)) then
      $conditions \leftarrow conditions \cup \{condition\}$
  /* Loop over the power-set of conditions */
  for $partition \in 2^{conditions}$ do
    $\psi_i \leftarrow \top$
    /* Conditions in the partition are true, others are false */
    for $clause \in conditions$ do
      if $clause \in partition$ then $\psi_i \leftarrow \psi_i \wedge clause$
                                else $\psi_i \leftarrow \psi_i \wedge \neg clause$
    $\Psi \leftarrow \Psi \cup \{\psi_i\}$
```

Partitioning the Query.

Algorithm 3 begins with the selection predicate ϕ in the shim query $\mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi)$, and outputs a set of fragments $\Psi = \{\psi_i\}$. Fragments are formed from the set of all possible truth assignments to a set of candidate conditions. Candidate conditions are obtained from if statements appearing in ϕ that have deterministic conditions, and that branch between deterministic and non-deterministic cases.

Example 4: Recall in Example 2, we generate a new C-Table using the VG-RA query. We now issue a query:

```
SELECT type FROM SaneProduct
WHERE brand = 'Apple' AND category = 'phone'
```

. The query has the non-deterministic condition (ϕ):

$$(\textbf{\textit{if}}\ brand\ \texttt{is null}\ \textbf{\textit{then}}\ Var('b', \texttt{ROWID})\ \textbf{\textit{else}}\ brand) = \text{‘}Apple'$$
$$\wedge\ (\textbf{\textit{if}}\ cat\ \texttt{is null}\ \textbf{\textit{then}}\ Var('c', \texttt{ROWID})\ \textbf{\textit{else}}\ cat) = \text{‘}phone'$$

There are two candidate conditions in ϕ: $brand$ `is null` and cat `is null`. Thus, Algorithm 3 creates 4 partitions: $\psi_1 = (\neg brand\ \texttt{is null} \wedge \neg cat\ \texttt{is null})$, $\psi_2 = (brand\ \texttt{is null} \wedge \neg cat\ \texttt{is null})$, $\psi_3 = (\neg brand\ \texttt{is null} \wedge cat\ \texttt{is null})$, and finally $\psi_4 = (brand\ \texttt{is null} \wedge cat\ \texttt{is null})$.

Segmenting ϕ. Using `isDet` from Algorithm 1, we partition the conjunctive terms of $\phi[\psi_i]$ into deterministic and non-deterministic components $\phi_{i,det}$ and $\phi_{i,var}$, respectively so that

$$(\phi_{i,det} \wedge \phi_{i,var}) \equiv \phi[\psi_i]$$

Note that Algorithm 3 may return a set of conditions that is not disjoint. We apply an additional check for overlap before using the output of this algorithm to partition a query.

Partitioning Complex Boolean Formulas.

We next describe a more aggressive partitioning strategy that uses the structure of ϕ to create partitions where each partition depends on exactly the same set of Var terms. To determine the set of partitions for each sub-query, we use a recursive traversal through the structure of ϕ, as shown in in Algorithm 4. The idea of the algorithm is that, in a fine-grained partition, there are exactly 2^N sub-queries union-ed together, where N is the number of atoms in where clause. For each subsets i (i from 1 to 2^N) of atoms, Algorithm 4 generates the condition ϕ_i and the corresponding selection predicate to select all rows having the same lineage.

Algorithm 4 generalPartition

```
In: $\phi$: A non-deterministic boolean expression considered as
    a tree structure, a set of atoms $\{a_i\}$
Out: $\Psi$: A set of partitions $\{\psi_i\}$ and corresponding conditions
    $\{\phi_i\}$
  if $\phi$ is a single atom then
    return
  if $\phi$.leftChild is an operator then
    generalPartition(root.leftchild)
  if $\phi$.rightChild is an operator then
    generalPartition(root.rightchild)
  visit($\phi$,$\{a_i\}$);
```

The partition approach makes full use of the backend database engine by splitting the query into deterministic and non-deterministic fragments. The lineage of the condition for each sub-query is simpler, and typically no longer data-dependent. As a consequence, explanation objects can be shared across all rows in the partition. The number of partitions obtained with both partitioning schemes is exponential in the number of candidate conditions. Partitions could conceivably be combined, increasing the number of redundant tuples processed by Mimir to create a lower-complexity query. In the extreme, we might have only two partitions: one deterministic and one non-deterministic. We leave the design of such a partition optimizer to future work.

V. Query Performance

In this section, we explore how specializing for narrower representations of uncertainty can improve performance.

Mimir [10] is an existing probabilistic database framework written in Scala. To provide an apples-to-apples comparison, we have modified Mimir to support each of the evaluation strategies listed above. All evaluation was performed on a 16-core 2.6 GHz Intel Xeon with 32GB of RAM, running RHEL 6.5, Oracle Java SE 1.8, and Scala 2.10. All experiments were

Algorithm 5 visit

```
In: $\phi$: A non-deterministic boolean expression considered as a tree structure,$\{a_i\}$
Out: $\Psi$: A set of partitions $\{\psi_i\}$ and corresponding conditions $\{\phi_i\}$
  if $\{a_i\}$ contains $\phi$.leftChild.$\phi_i$ and
  $\{a_i\}$ contains $\phi$.rightChild.$\phi_i$ then
    $\phi_i$.combine($\phi$.leftChild.$\phi_i$,$\phi$.rightChild.$\phi_i$);
    $\psi_i$.add($\phi$.leftChild.$\psi_i$);
    $\psi_i$.add($\phi$.rightChild.$\psi_i$);
  else
    if $\{a_i\}$ contains $\phi$.leftChild.$\phi_i$ then
      $\phi_i$.add($\phi$.leftChild.$\phi_i$);
      if $\phi$ is instanceOf OR Operator then
        $\psi_i$.add(NOT $\phi$.rightChild);
      if $\phi$ is instanceOf AND Operator then
        $\psi_i$.add($\phi$.rightChild);
    else
      if $\{a_i\}$ contains $\phi$.rightChild.$\phi_i$ then
        $\phi_i$.add($\phi$.rightChild.$\phi_i$);
        if $\phi$ is instanceOf OR Operator then
          $\psi_i$.add(NOT $\phi$.leftChild);
        if $\phi$ is instanceOf AND Operator then
          $\psi_i$.add($\phi$.leftChild);
      else
        $\psi_i$.add($\phi$);
```

Strategy	Q1	Q2	Q3
Inline	8.2s	55.2s	9.8s
TupleBundle	85.5s	676.6s	103.3s
Partition	>1hr	739.7s	>1hr

Fig. 11: **Evaluation Strategy Performance on PDBench**

performed with a warm cache and single-threaded. As a performance measure, we used PDBench [4], an adaptation of the classical TPC-H [20] benchmark for probabilistic databases. PDBench generates uncertainty in the form of functional dependency violations throughout the seven TPC-H tables. Our evaluation is based on the three queries specified in the PDBench benchmark, variations of TPC-H Queries 3, 6, and 7, respectively. We used default PDBench uncertainty settings and a Scaling Factor of 1 (1GB of raw data).

Raw data was stored as a single, bulk table rather than the columnar representation used by PDBench. For the Customer, Orders, Supplier, and Nation tables, we created a BTree index on the table's key attribute and used Mimir's functional dependency repair lens operator[5] to repair the table's key attribute. We repaired and indexed the Lineitem table on a new column of unique tuple identifiers generated by PDBench.

Our results are shown in Figure 11. **Inline** and **Partition** denote the three algorithms described in Section III. **TupleBundle** denotes an approach resembling the TupleBundles of MCDB [3] with a bundle size of 10. Times reported include only time taken to process the query itself. We also use a pre-processing step in which each input table is analyzed for FD violations. Although times for this step are unreported, all input tables completed the analysis within half an hour using a naive implementation without optimization.

[5]This operator replicates the functionality of the probabilistic repair-key operator in MayBMS

VI. Related Work

Probabilistic Databases. The field of probabilistic databases [1] explores query processing over ambiguous or uncertain data. Numerous probabilistic database systems have arisen, including MYSTIQ [14], Trio [15], MayBMS/Sprout [13], [16], Pip [12], Orion 2.0 [17], MCDB [3], and Mimir [9], [10]. Of these, only Mimir considers user interactions with probabilistic query results, and then only informally. This paper differentiates itself by taking user-focused approach to the design of a probabilistic database engine.

Data Uncertainty. Uncertainty or ambiguity in data arises in a variety of contexts. A prominent example is low-quality data [6], [21], [22], where the emphasis is on incomplete data curation tasks [7], [10], [23], or outliers in the data [24]. Uncertainty, incompleteness, and ambiguity have also been explored in other data management contexts, including distributed systems [25], approximate query processing [26], [27], semi-structured data [28], [29], querying models [30], [31], and novel query interfaces [32], [33].

Uncertainty in Database Interfaces. Specific work from these domains to establish the importance of visualization and interactivity includes online aggregation (OLA) tools [8] which presents a variety of interactions through a graphical user interface, including the reading of an intuitive ”completion meter”, access to precise statistical bounds on presented results, and the option to halt progress at any time. More recently, several tools have explored visualizations for expressing ambiguity in a data set while also prioritizing interactivity and easily accessible data provenance. Wrangler [21] is an Interactive Visual Specification of Data Transformation Types that identifies errors by flagging inconsistent data types and inferred semantic roles. Wrangler communicates this information by color and textual suggestion, representing and explaining different levels of denigration from a complete verification. The GestureQuery [32] system allows users to specify joins by dragging two tables together. This initially creates an underspecified join; the system combines a probabilistic model of join quality with feedback about system interpretation and an easily accessible preview of join results. MCDB [3] is a tool that uses the Monte Carlo approach to query uncertain data. The MCDB interface presents results through classical statistical metrics such as expectation and standard deviation, as well as more visual representations like histograms. Jigsaw [34] is a dashboard for parameter-space exploration over modeled scenarios. It generates graphical representations using box-plots and an OLA-style progress meter, providing a visual indication of parts of the parameter space that can be interacted with immediately. Although these systems each employ specific representations of uncertainty, the representations are used in service of a different goal (e.g., quick query responses or data cleaning interactions).

To the best of our knowledge, ours is the first effort to explore how users are affected by perceived uncertainty in relational data. This work extends and expands on our preliminary study [11]. As already noted, the study presented in this paper addresses many of the limitations of our prior study by examining broader representation among study participants, presenting additional representations of uncertainty, and randomizing the presentation order.

Operations Research. Perception of uncertainty in data has also been explored in other fields, most notably in Operations Research [35], where decision-making must take into account incomplete circumstantial knowledge. Efforts in this space are often specialized and implemented at the application-layer. To the best of our knowledge, ours is the first to explore the design of a DBMS that incorporates the effect of uncertain data on human interactivity.

VII. Conclusion

In this paper, we addressed the need for uncertain and probabilistic data management systems to interact directly with users. We outlined a user study that showed how users could be convinced to incorporate uncertainty into decisions based on imperfect data or query results. As a result of this study, we showed that users made rational decisions more quickly with low-bandwidth uncertainty representations like red text or red backgrounds. In contrast to the more extensive derivation of probability distributions performed by classical probabilistic databases, such representations can be constructed quickly and efficiently. We outlined and evaluated a range of strategies for constructing low-bandwidth representations, and showed that they can outperform classical probabilistic databases. This work represents one of the first steps in the design of a user-focused probabilistic database system. Future work includes exploring interfaces other forms of uncertainty like row-level and result incompleteness as well as interfaces for helping users to explore, debug, and repair uncertainty in query results.

Acknowledgements. This work was supported by a gift from Oracle, NSF awards #1409551 and #1640864, and NPS award #N00244-16-1-0022. Any views and opinions presented are solely those of the authors and not of Oracle, the National Science Foundation, or the Naval Postgraduate School.

References

[1] D. Suciu, D. Olteanu, C. Ré, and C. Koch, "Probabilistic databases," *Synthesis Lectures on Data Management*, vol. 3, no. 2, pp. 1–180, 2011.

[2] R. Fink, J. Huang, and D. Olteanu, "Anytime approximation in probabilistic databases," *VLDB J.*, vol. 22, no. 6, pp. 823–848, 2013. [Online]. Available: http://dx.doi.org/10.1007/s00778-013-0310-5

[3] 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*, 2008.

[4] L. Antova, T. Jansen, C. Koch, and D. Olteanu, "Fast and simple relational processing of uncertain data," in *ICDE*, April 2008, pp. 983–992.

[5] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller, "Crowdsourced databases: Query processing with people." CIDR, 2011.

[6] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye, "Katara: A data cleaning system powered by knowledge bases and crowdsourcing," in *SIGMOD*, 2015.

[7] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy, "Pay-as-you-go user feedback for dataspace systems," in *SIGMOD*, 2008.

[8] J. M. Hellerstein, P. J. Haas, and H. J. Wang, "Online aggregation," in *SIGMOD*, 1997.

[9] Y. Yang, "On-demand query result cleaning," in *VLDB PhD Workshop*, 2014.

[10] Y. Yang, N. Meneghetti, R. Fehling, Z. H. Liu, and O. Kennedy, "Lenses: An on-demand approach to etl," *PVLDB*, vol. 8, no. 12, pp. 1578–1589, 2015.

[11] P. Kumari, S. Achmiz, and O. Kennedy, "Communicating data quality in on-demand curation," in *QDB*, 2016.

[12] O. Kennedy and C. Koch, "PIP: A database system for great and small expectations," in *ICDE*, 2010.

[13] R. Fink, A. Hogue, D. Olteanu, and S. Rath, "Sprout^2: a squared query engine for uncertain web data," in *SIGMOD*, 2011.

[14] 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*, 2005.

[15] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. U. Nabar, T. Sugihara, and J. Widom, "Trio: A system for data, uncertainty, and lineage," in *VLDB*, 2006.

[16] L. Antova, C. Koch, and D. Olteanu, "$10^{(10^6)}$ worlds and beyond: efficient representation and processing of incomplete information," *VLDB J.*, vol. 18, no. 5, pp. 1021–1040, 2009.

[17] S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. Hambrusch, and R. Shah, "Orion 2.0: Native support for uncertain data," in *SIGMOD*, 2008.

[18] T. Imielinski and W. L. Jr., "Incomplete information in relational databases," *J. ACM*, vol. 31, no. 4, pp. 761–791, 1984.

[19] T. J. Green, G. Karvounarakis, and V. Tannen, "Provenance semirings," in *PODS*, 2007.

[20] T. P. P. Council, "TPC-H specification," http://www.tpc.org/tpch/.

[21] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer, "Wrangler: Interactive visual specification of data transformation scripts," in *SIGCHI*, 2011.

[22] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo, "A sample-and-clean framework for fast and accurate query processing on dirty data," in *SIGMOD*, 2014.

[23] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas, "Guided data repair," *PVLDB*, vol. 4, no. 5, pp. 279–289, 2011.

[24] F. Chirigati, H. Doraiswamy, T. Damoulas, and J. Freire, "Data polygamy: the many-many relationships among urban spatio-temporal data sets," in *SIGMOD*, 2016.

[25] W. Lang, R. V. Nehme, E. Robinson, and J. F. Naughton, "Partial results in database systems," in *SIGMOD*, 2014.

[26] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica, "BlinkDB: Queries with bounded errors and bounded response times on very large data," in *EuroSys*, 2013.

[27] S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. Jordan, S. Madden, B. Mozafari, and I. Stoica, "Knowing when you're wrong: Building fast and reliable approximate query processing systems," in *SIGMOD*, 2014.

[28] M. DiScala and D. J. Abadi, "Automatic generation of normalized relational schemas from nested key-value data," in *SIGMOD*, 2016.

[29] W. Spoth, B. S. Arab, E. S. Chan, D. Gawlick, A. Ghoneimy, B. Glavic, B. Hammerschmidt, O. Kennedy, S. Lee, Z. H. Liu, X. Niu, and Y. Yang, "Adaptive schema databases," in *CIDR*, 2017.

[30] 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.

[31] A. Deshpande and S. Madden, "MauveDB: supporting model-based user views in database systems," in *SIGMOD*, 2006.

[32] L. Jiang, M. Mandel, and A. Nandi, "GestureQuery: A multitouch database query interface," *PVLDB*, vol. 6, no. 12, pp. 1342–1345, 2013.

[33] F. Li and H. V. Jagadish, "Nalir: An interactive natural language interface for querying relational databases," in *SIGMOD*, 2014.

[34] O. Kennedy and S. Nath, "Jigsaw: efficient optimization over uncertain enterprise data," in *SIGMOD*, 2011.

[35] A. M. Bisantz, R. Finger, Y. Seong, and J. Llinas, "Human performance and data fusion based decision aids," in *FUSION*, 1999.