10^{10^6} Worlds and Beyond: Efficient Representation and Processing of Incomplete Information

Lyublena Antova, Christoph Koch, and Dan Olteanu

Lehrstuhl für Informationssysteme
Universität des Saarlandes, Germany
{lublena,koch,olteanu}@infosys.uni-sb.de

Abstract

We present a decomposition-based approach to managing incomplete information. We introduce world-set decompositions (WSDs), *a space-efficient and complete representation system for finite sets of worlds. We study the problem of efficiently evaluating relational algebra queries on world-sets represented by WSDs. We also evaluate our technique experimentally in a large census data scenario and show that it is both scalable and efficient.*

1 Introduction

Incomplete information is commonplace in real-world databases. Classical examples can be found in data integration and wrapping applications, linguistic collections, or whenever information is manually entered and is therefore prone to inaccuracy or partiality.

There has been little research so far into expressive *yet scalable* systems for representing incomplete information. Current techniques can be classified into two groups. The first group includes representation systems such as *v-tables* [14] and *or-set relations* [15] which are not strong enough to represent the result of any relational algebra query within the same formalism. In v-tables the tuples can contain both constants and variables, and each combination of possible values for the variables yields a possible world. Relations with or-sets can be viewed as v-tables, where each variable occurs only at a single position in the table and can only take values from a fixed finite set, the or-set of the field occupied by the variable. The so-called *c-tables* [14] belong to the second group of formalisms. They extend v-tables with conditions specified by logical formulas over the variables, thus constraining the possible values. Although c-tables are a strong representation system, they have not found application in practice. The main reason for this is probably that managing c-tables directly is rather inefficient. Even very basic problems such as deciding whether a tuple is in at least one world represented by the c-table are intractable [2].

As a motivation, consider two manually completed forms that may originate from a census and which allow for more than one interpretation (Figure 1). For simplicity we assume that social security numbers consist of only three digits. For instance, Smith's social security number can be read either as "185" or as "785". We can represent the available information using a relation with or-sets:

(TID)	S	N	M
t_1	{ 185, 785 }	Smith	{ 1, 2 }
t_2	{ 185, 186 }	Brown	{ 1, 2, 3, 4 }

It is easy to see that this or-set relation represents $2 \cdot 2 \cdot 2 \cdot 4 = 32$ *possible worlds*.

Given such an incompletely specified database, it must of course be possible to access and process the data. Two data management tasks shall be pointed out as particularly important, the evaluation of queries on the data and *data cleaning* [16, 13, 17], by which certain worlds can be shown to be impossible and can be excluded. The results of both types of operation turn out not to be representable by or-set relations in general. Consider for example the integrity constraint that all social security numbers be unique. For our example database, this constraint excludes 8 of the 32 worlds, namely those in which both tuples have the value 185 as social security number. It is impossible to represent the remaining 24 worlds using or-set relations. This is an example of a constraint that can be used for data cleaning; similar problems are observed with queries, e.g., the query asking for pairs of persons with differing social security numbers.

What we could do is store each world explicitly using a table called a *world-set relation* of a given set of worlds. Each tuple in this table represents one world and is the concatenation of all tuples in that world (see Figure 2).

The most striking problem of world-set relations is their size. If we conduct a survey of 50 questions on a population of 200 million and we assume that one in 10^4 answers can be read in just two different ways, we get 2^{10^6} worlds. Each such world is a substantial table of 50 columns and $2 \cdot 10^8$ rows. We cannot store all these worlds explicitly in

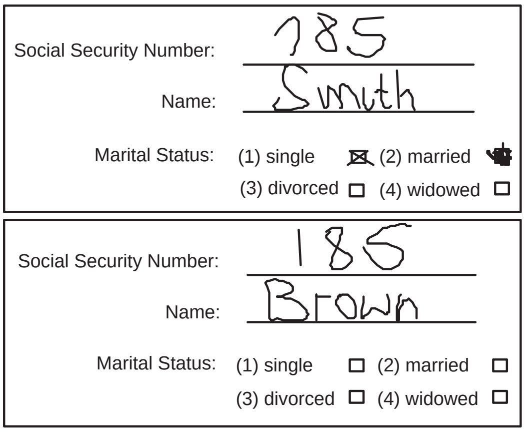


Figure 1. Two completed survey forms.

$t_1.S$	$t_1.N$	$t_1.M$	$t_2.S$	$t_2.N$	$t_2.M$
185	Smith	1	186	Brown	1
185	Smith	1	186	Brown	2
185	Smith	1	186	Brown	3
185	Smith	1	186	Brown	4
185	Smith	2	186	Brown	1
		⋮			
785	Smith	2	186	Brown	4

Figure 2. World-set relation containing only worlds with unique social security numbers.

a world-set relation (which would have 10^{10} columns and 2^{10^6} rows). Data cleaning will often eliminate only some of these worlds, so a DBMS should manage those that remain.

This paper aims at dealing with this complexity and proposes the new notion of *world-set decompositions (WSDs)*. These are decompositions of a world-set relation into several relations such that their product (using the product operation of relational algebra) is again the world-set relation.

Example 1.1 The world-set represented by our initial or-set relation can also be represented by the product

$t_1.S$
185
785

×

$t_1.N$
Smith

×

$t_1.M$
1
2

×

$t_2.S$
185
186

×

$t_2.N$
Brown

×

$t_2.M$
1
2
3
4

Example 1.2 In the same way we can represent the result of data cleaning with the uniqueness constraint for the social security numbers as the product of Figure 3.

$t_1.S$	$t_2.S$
185	186
785	185
785	186

×

$t_1.N$
Smith

×

$t_1.M$
1
2

×

$t_2.N$
Brown

×

$t_2.M$
1
2
3
4

Figure 3. WSD of the relation in Figure 2.

One can observe that the result of this product is exactly the world-set relation in Figure 2. The presented decomposition is based on the *independence* between sets of fields, subsequently called *components*. Only fields that depend on each other, for example $t_1.S$ and $t_2.S$, belong to the same component. Since $\{t_1.S, t_2.S\}$ and $\{t_1.M\}$ are independent, they are put into different components. □

In practice, it is often the case that fields or even tuples carry the same values in all worlds. For instance, in the census data scenario discussed above, we assumed that only one field in 10000 has several possible values. Such a world-set decomposes into a WSD in which most fields are in component relations that have precisely one tuple.

We will also consider a refinement of WSDs, *WSDTs*, which store information that is the same in all possible worlds once and for all in so-called *template relations*.

Example 1.3 The world-set of the previous examples can be represented by the WSDT of Figure 4. □

Template	S	N	M
t_1	?	Smith	?
t_2	?	Brown	?

t_1.S	t_2.S
185	186
785	185
785	186

×

t_1.M
1
2

×

t_2.M
1
2
3
4

Figure 4. WSD with a template relation.

WSDTs combine the advantages of WSDs and c-tables. In fact, WSDTs can be naturally viewed as c-tables whose formulas have been put into a *normal form* represented by the component relations, and null values '?' in the template relations represent fields on which the worlds disagree. Indeed, each tuple in the product of the component relations is a possible value assignment for the variables in the template relation. The following c-table with global condition Φ is equivalent to the WSDT in Figure 4.

T	S	N	M
	x	Smith	y
	z	Brown	w

$$\begin{aligned}\Phi \quad = \quad & ((x = 185 \land z = 186) \lor (x = 785 \land z = 185) \lor \\ & (x = 785 \land z = 186)) \land (y = 1 \lor y = 2) \land \\ & (w = 1 \lor w = 2 \lor w = 3 \lor w = 4)\end{aligned}$$

The technical contributions of this paper are as follows.

- We formally introduce WSDs and WSDTs and study some of their properties. Our notion is a refinement of the one presented above and allows to represent worlds over multi-relation schemas which contain relations with varying numbers of tuples. WSD(T)s can represent any finite set of possible worlds over relational databases and are therefore a strong representation system for *any relational query language*.
- A practical problem with WSDs and WSDTs is that a DBMS that manages such representations has to support relations of arbitrary arity: the schemata of the component relations of a decomposition depend on the

data. Unfortunately, DBMS (e.g. PostgreSQL) in practice often do not support relations beyond a fixed arity.

For that reason we present refinements of the notion of WSDs, the *uniform WSDs (UWSDs)*, and their extension by template relations, the *UWSDTs*, and study their properties as representation systems.

- We show how to process relational algebra queries over world-sets represented by UWSDTs. For illustration purposes, we discuss query evaluation in the context of the much more graphic WSDs.

  We also develop a number of optimizations and techniques for normalizing the data representations obtained by queries to support scalable query processing even on very large world-sets.
- We describe a prototype implementation built on top of the PostgreSQL RDBMS. Our system is called MayBMS and supports the management of incomplete information using UWSDTs.
- We report on our experimental evaluation of UWSDTs as a representation system for large finite sets of possible worlds. Our experiments show that UWSDTs allow highly scalable techniques for managing incomplete information. We found that the size of UWSDTs obtained as query answers or data cleaning results remains close to that of a single world. Furthermore, the processing time for queries on UWSDTs is also comparable to processing just a single world and thus a classical relational database.
- For our experiments, we develop data cleaning techniques in the context of UWSDTs. To clean data of inconsistent worlds we chase a set of equality-generating dependencies on UWSDTs, which we briefly describe.

WSDs are designed to cope with large sets of worlds, which exhibit local dependencies and large commonalities. Note that this data pattern can be found in many applications. Besides the census scenario, our technical report [4] describes two further applications: managing inconsistent databases using minimal repairs [7, 9] and medicine data.

A fundamental assumption of this work is that one wants to manage *finite sets of possible worlds*. This is justified by previous work on representation systems starting with Imielinski and Lipski [14], by recent work [12, 3, 8], and by current application requirements. Our approach can deal with databases with unresolved uncertainties. Such databases are still valuable. It should be possible to do data transformations that preserve as much information as possible, thus necessarily mapping between sets of possible worlds. In this sense, WSDs represent a *compositional framework* for querying and data cleaning. A different approach is followed in, e.g., [7, 10], where the focus is on finding *certain answers* of queries on incomplete and inconsistent databases.

Related Work. The probabilistic databases of [12, 11] and the dirty relations of [3] are examples of practical representation systems that are not strong for relational algebra. Such formalisms close the possible worlds semantics using clean answers [3] and probabilistic-ranked retrieval [12]. We show in [4] how a simple probabilistic WSD extension can represent the probabilistic databases of [12] and the dirty relations of [3].

In parallel to our approach, [19, 8] propose ULDBs that combine uncertainty and a low-level form of lineage to model any finite world-set. Like the dirty relations of [3], ULDBs represent a set of independent tuples with alternatives. Lineage is then used to represent dependencies among alternatives of different tuples and thus is essential for the expressive power of the formalism.

As both ULDBs and WSDs can model any finite world-set, they inherently share some similarities, yet differ in important aspects. WSDs support efficient algorithms for finding a minimal data representation based on relational factorization. Differently from ULDBs, WSDs allow dependencies at the level of tuple fields, not only of tuples. This causes, for instance, or-set relations to have linear representations as WSDs, but (in general) exponential representations as ULDBs. As reported in [8], resolving tuple dependencies, i.e., tracking which alternatives of different tuples belong to the same world, often requires to compute expensive lineage closure. Additionally, query operations on ULDBs can produce inconsistencies and anomalies, such as erroneous dependencies and inexistent tuples. In contrast, WSDs share neither of these pitfalls. As no implementation of ULDBs was available at the time of writing this document, no experimental comparison of ULDBs and WSDs could be established. ULDB encodings of the examples in this section are given at the MayBMS project homepage [5].

2 Preliminaries

We use the named perspective of the relational model with the operations selection σ, projection π, product $\times$, union $\cup$, difference $-$, and attribute renaming δ (cf. e.g. [1]). A *relational schema* is a tuple $\Sigma = (R_1[U_1], \ldots, R_k[U_k])$, where each R_i is a relation name and U_i is a set of attribute names. Let $\mathbf{D}$ be a set of domain elements. A *relation* over schema $R[A_1, \ldots, A_k]$ is a set of tuples $(A_1 : a_1, \ldots, A_k : a_k)$ where $a_1, \ldots, a_k \in \mathbf{D}$. A *relational database* $\mathcal{A}$ over schema Σ is a set of relations $R^{\mathcal{A}}$, one for each relation schema $R[U]$ from Σ. Sometimes, when no confusion of database may occur, we will use R rather than $R^{\mathcal{A}}$ to denote one particular relation over schema $R[U]$. By the size of a relation R, denoted $|R|$, we refer to the number of tuples in R. For a relation R over schema $R[U]$, let $\mathcal{S}(R)$ denote the set U of its attributes and let $ar(R)$ denote the arity of R.

A *product* m*-decomposition* of a relation R is a set of non-nullary relations $\{C_1, \ldots, C_m\}$ such that $C_1 \times \cdots \times C_m = R$. The relations $C_1, \ldots, C_m$ are called *components*. A product m-decomposition of R is *maximal* if there is no product n-decomposition of R with $n > m$.

A set of *possible worlds* (or *world-set*) over schema Σ is a set of databases over schema Σ. Let $\mathbf{W}$ be a set of structures, rep be a function that maps from $\mathbf{W}$ to world-sets of the same schema. Then $(\mathbf{W}, rep)$ is a *strong representation system* for a query language if, for each query Q of that language and each $\mathcal{W} \in \mathbf{W}$ such that Q is applicable to the worlds in $rep(\mathcal{W})$, there is a structure $\mathcal{W}' \in \mathbf{W}$ such that $rep(\mathcal{W}') = \{Q(\mathcal{A}) \mid \mathcal{A} \in rep(\mathcal{W})\}$. Obviously,

Lemma 2.1 *If rep is a function from a set of structures $\mathbf{W}$ to the set of all finite world-sets, then $(\mathbf{W}, rep)$ is a strong representation system for any relational query language.*

3 World-Set Decompositions

In order to use classical database techniques for storing and querying incomplete data, we develop a scheme for representing a world-set $\mathbf{A}$ by a single relational database.

Let $\mathbf{A}$ be a finite world-set over schema $\Sigma = (R_1, \ldots, R_k)$. For each R in Σ, let $|R|_{\max} = \max\{|R^{\mathcal{A}}| : \mathcal{A} \in \mathbf{A}\}$ denote the maximum cardinality of relation R in any world of $\mathbf{A}$. Given a world $\mathcal{A}$ with $R^{\mathcal{A}} = \{t_1, \ldots, t_{|R^{\mathcal{A}}|}\}$, let $t_{R^{\mathcal{A}}}$ be the tuple obtained as the concatenation (denoted $\circ$) of the tuples of $R^{\mathcal{A}}$ in an arbitrary order padded with a special null value $\perp \notin \mathbf{D}$ up to arity $ar(R) \cdot |R|_{\max}$,

$$t_{R^{\mathcal{A}}} := t_1 \circ \cdots \circ t_{|R^{\mathcal{A}}|} \circ (\underbrace{\perp, \ldots\ldots\ldots\ldots, \perp}_{ar(R) \cdot (|R|_{\max} - |R^{\mathcal{A}}|)}).$$

Then tuple $t_{\mathcal{A}} := t_{R_1^{\mathcal{A}}} \circ \cdots \circ t_{R_k^{\mathcal{A}}}$ encodes all the information in world $\mathcal{A}$. The "dummy" tuples with $\perp$-values are only used to ensure that the relation R has the same number of tuples in all worlds in $\mathbf{A}$.

By a *world-set relation* of a world-set $\mathbf{A}$, we denote the relation $\{t_{\mathcal{A}} \mid \mathcal{A} \in \mathbf{A}\}$. This world-set relation has schema $\{R.t_i.A_j \mid R[U] \in \Sigma, 1 \leq i \leq |R|_{\max}, A_j \in U\}$. Note that in defining this schema we use t_i to denote the position (or identifier) of tuple t_i in $t_{R^{\mathcal{A}}}$ and not its value.

Given the above definition that canonically turned every world in a tuple of a world-set relation, computing the initial world-set is an easy exercise. In order to have every world-set relation define a world-set, let a tuple extracted from some $t_{R^{\mathcal{A}}}$ be in $R^{\mathcal{A}}$ iff it does not contain any occurrence of the special symbol $\perp$. That is, we map $t_{R^{\mathcal{A}}} = (a_1, \ldots, a_{ar(R) \cdot |R|_{\max}})$ to $R^{\mathcal{A}}$ as

$$t_{R^{\mathcal{A}}} \mapsto \{(a_{ar(R) \cdot k+1}, \ldots, a_{ar(R) \cdot (k+1)}) \mid 0 \leq k < |R|_{\max}, \\ a_{ar(R) \cdot k+1} \neq \perp, \ldots, a_{ar(R) \cdot (k+1)} \neq \perp\}.$$

Observe that although world-set relations are not unique as we have left open the ordering in which the tuples of a given world are concatenated, all world-set relations of a world-set $\mathbf{A}$ are equally good for our purposes because they can be mapped invariantly back to $\mathbf{A}$. Note that for each world-set relation a maximal decomposition exists, is unique, and can be efficiently computed [6].

Definition 3.1 Let $\mathbf{A}$ be a world-set and W a world-set relation representing $\mathbf{A}$. Then a *world-set m-decomposition (m-WSD)* of $\mathbf{A}$ is a product m-decomposition of W.

Somewhat simplified examples of world-set relations and WSDs over a single relation R (thus "R" was omitted from the attribute names of the world-set relations) were given in Section 1. Further examples can be found in Section 4. It should be emphasized that with WSDs we can also represent multiple relational schemata and even components with fields from different relations.

It immediately follows from our definitions that

Proposition 3.2 *Any finite set of possible worlds can be represented as a world-set relation and as a 1-WSD.*

Corollary 3.3 (Lemma 2.1) *WSDs are a strong representation system for any relational query language.*

As pointed out in Section 1, this is not true for or-set relations. For the relatively small class of world-sets that can be represented as or-set relations, the size of our representation system is linear in the size of the or-set relations. As seen in the examples, our representation is *much more space-efficient than world-set relations*.

Adding Template Relations. We now present our refinement of WSDs with so-called *template relations*. A template stores information that is the same in all possible worlds and contains special values '?' $\notin \mathbf{D}$ in fields at which different worlds disagree.

Let $\Sigma = (R_1, \ldots, R_k)$ be a schema and $\mathbf{A}$ a finite set of possible worlds over Σ. Then, the database $(R_1^0, \ldots, R_k^0, \{C_1, \ldots, C_m\})$ is called an *m-WSD with template relations (m-WSDT)* of $\mathbf{A}$ iff there is a WSD $\{C_1, \ldots, C_m, D_1, \ldots, D_n\}$ of $\mathbf{A}$ such that $|D_i| = 1$ for all i and if relation D_i has attribute $R_j.t.A$ and value v in its unique $R_j.t.A$-field, then the template relation R_j^0 has a tuple with identifier t whose A-field has value v.

Of course WSDTs again can represent any finite world-set and are thus a strong representation system for any relational query language. Example 1.3 shows a WSDT for the running example of the introduction.

Uniform World-Set Decompositions. In practice database systems often do not support relations of arbitrary arity (e.g., WSD components). For that reason we introduce next a modified representation of WSDs called *uniform*

R^0	S	N	M
t_1	?	Smith	?
t_2	?	Brown	3

C	FID	LWID	VAL
	(R, t_1, S)	1	185
	(R, t_2, S)	1	186
	(R, t_1, S)	2	785
	(R, t_2, S)	2	185
	(R, t_1, S)	3	785
	(R, t_2, S)	3	186
	(R, t_1, M)	1	1
	(R, t_1, M)	2	2

F	FID	CID
	(R, t_1, S)	C_1
	(R, t_1, M)	C_2
	(R, t_2, S)	C_1

W	CID	LWID
	C_1	1
	C_1	2
	C_1	3
	C_2	1
	C_2	2

Figure 5. A UWSDT corresponding to the WSDT of Figure 4.

WSDs. Instead of having a variable number of component relations, possibly with different arities, we store all values in a single relation C that has a fixed schema. We use the fixed schema consisting of the three relation schemata $C[FID, LWID, VAL], F[FID, CID], W[CID, LWID]$, where FID is a triple[1] $(Rel, TupleID, Attr)$ denoting the $Attr$-field of tuple $TupleID$ in database relation Rel.

Given a WSD $\{C_1, \ldots, C_m\}$ with schemata $C_i[U_i]$, we populate the corresponding UWSD as follows.

- $((R, t, A), s, v) \in C$ iff, for some (unique) i, $R.t.A \in U_i$ and the field of column $R.t.A$ in the tuple with id s of C_i has value v.
- $F := \{((R, t, A), C_i) \mid 1 \leq i \leq m,\ R.t.A \in U_i\}$,
- $(C_i, s) \in W$ iff there is a tuple with identifier s in C_i.

Intuitively, the relation C stores each value from a component together with its corresponding field identifier and the identifier of the component-tuple in the initial WSD (column $LWID$ of C). The relation F contains the mapping between tuple fields and component identifiers, and W keeps track of the worlds present for a given component.

In general, the VAL column in the component relation C must store values for fields of different type. One possibility is to store all values as strings and use casts when required. Alternatively, one could have one component relation for each data type. In both cases the schema remains fixed.

Finally, we add template relations to UWSDs in complete analogy with WSDTs, thus obtaining the UWSDTs.

Example 3.4 We modify the world-set represented in Figure 3 such that the marital status in t_2 can only have the value 3. Figure 5 is then the uniform version of the WSDT of Figure 4. Here R^0 contains the values that are the same in all worlds. For each field that can have more than one possible value, R^0 contains a special placeholder, denoted by '?'. The possible values for the placeholders are defined in the component table C. In practice, we can expect that the majority of the data fields can take only one value across all worlds, and can be stored in the template relation. □

Proposition 3.5 *Any finite set of possible worlds can be represented as a* 1*-UWSD and as a* 1*-UWSDT.*

It follows again that UWSD(T)s are a strong representation system for *any relational query language*.

4 Queries on Decompositions

In this section we study the query evaluation problem for WSDs. As pointed out before, UWSDTs are a better representation system than WSDs; nevertheless WSDs are simpler to explain and visualize and the main issues regarding query evaluation are the same for both systems.

The goal of this section is to provide, for each relational algebra query Q, a query $\hat{Q}$ such that for a WSD $\mathcal{W}$,

$$rep(\hat{Q}(\mathcal{W})) = \{Q(\mathcal{A}) \mid \mathcal{A} \in rep(\mathcal{W})\}.$$

Of course we want to evaluate queries directly on WSDs using $\hat{Q}$ rather than process the individual worlds using Q.

When compared to traditional query evaluation, the evaluation of relational queries on WSDs poses new challenges. First, since decompositions in general consist of several components, a query $\hat{Q}$ that maps from one WSD to another must be expressed as a set of queries, each of which defines a different component of the output WSD. Second, as certain query operations may cause new dependencies between components to develop, some components may have to be merged (i.e., part of the decomposition undone using the product operation $\times$). Third, the answer to a (sub)query Q_0 must be represented within the same decomposition as the input relations; indeed, we want to compute a decomposition of world set $\{(\mathcal{A}, Q_0(\mathcal{A})) \mid \mathcal{A} \in rep(\mathcal{W})\}$ in order to be able to resort to the input relations as well as the result of Q_0 within each world. Consider for example a query $\sigma_{A=1}(R) \cup \sigma_{B=2}(R)$. If we first compute $\sigma_{A=1}(R)$, we must not replace R by $\sigma_{A=1}(R)$, otherwise R will not be available for the computation of $\sigma_{B=2}(R)$. On the other hand, if $\sigma_{A=1}(R)$ is stored in a separate WSD, the connection between worlds of R and the selection $\sigma_{A=1}$ is lost and we can again not compute $\sigma_{A=1}(R) \cup \sigma_{B=2}(R)$.

We say that a relation P is a copy of another relation R in a WSD if R and P have the same tuples in every world represented by the WSD. For a component C, an attribute $R.t.A_i$ of C and a new attribute $P.t.B$, the function ext extends C by a new column $P.t.B$ that is a copy of $R.t.A_i$:

$$\mathsf{ext}(C, A_i, B) := \{(A_1 : a_1, \ldots, A_n : a_n, B : a_i) \mid (A_1 : a_1, \ldots, A_n : a_n) \in C\}$$

[1] That is, FID really takes three columns, but for readability we keep them together under a common name in this section.

Then $\mathsf{copy}(R, P)$ executes $C := \mathsf{ext}(C, R.t_i.A, P.t_i.A)$ for each component C and each $R.t_i.A \in \mathcal{S}(C)$.

Figure 6 presents implementations of the relational algebra operations selection (of the form $\sigma_{A\theta c}$ or $\sigma_{A\theta B}$, where A and B are attributes, c is a constant, and θ is a comparison operation, $=, \neq, <, \leq, >$, or $\geq$), and relational product on WSDs. In each case, the input WSD is *extended* by the result of the operation. The operations projection, union, difference, and renaming are defined in [4].

Let us now have a closer look at the evaluation of relational algebra operations on WSDs. For this, we use as running example the 7-WSD of Figure 7 representing a set of eight worlds over the relation R. Because of space limitations and our attempt to keep the WSDs readable, we consistently show in the following examples only the WSDs of the result relations.

```
algorithm select[Aθc]     // compute P := σ_{Aθc} R
begin
  copy(R, P);
  for each 1 ≤ i ≤ |P|_max do begin
    let C be the component of P.t_i.A;
    for each t_C ∈ C do
      if not (t_C.(P.t_i.A) θ c) then begin
        t_C.(P.t_i.A) := ⊥
        for each A' such that P.t_i.A' ∈ S(C) do
          t_C.(P.t_i.A') := ⊥;
      end
  end
end

algorithm select[AθB]     // compute P := σ_{AθB} R
begin
  copy(R, P);
  for each 1 ≤ i ≤ |P|_max do begin
    let C be the component of P.t_i.A;
    let C' be the component of P.t_i.B;
    if (C ≠ C') then
      replace components C, C' by C := C × C';
    for each t_C ∈ C do
      if not (t_C.(P.t_i.A) θ t_C.(P.t_i.B)) then begin
        t_C.(P.t_i.A) := ⊥
        for each A' such that P.t_i.A' ∈ S(C) do
          t_C.(P.t_i.A') := ⊥;
      end
  end
end

algorithm product     // compute T := R × S
begin
  for each 1 ≤ j ≤ |S|_max and component C do begin
    for each R.t_i.A ∈ S(C) do
      C := ext(C, R.t_i.A, T.t_ij.A);
  end;
  for each 1 ≤ i ≤ |R|_max and component C do begin
    for each S.t_j.A ∈ S(C) do
      C' := ext(C', S.t_j.A, T.t_ij.A);
  end
end
```

Figure 6. Evaluating selection and product operations on WSDs.

Selection with condition $A\theta c$. In order to compute a selection $P := \sigma_{A\theta c}(R)$, we first compute a copy P of relation R and subsequently drop tuples of P that do not match the selection condition.

Dropping tuples is a fairly subtle operation, since tuples can spread over several components and a component can define values for more than one tuple.

Thus a selection must not delete tuples from component relations, but should mark fields as belonging to deleted tuples using the special value $\perp$. To evaluate $\sigma_{A\theta c}(R)$, our selection algorithm of Figure 6 checks for each tuple t_i in the relation P and t_C in component C with attribute $P.t_i.A$ whether $t_C.(P.t_i.A)\theta c$. In the negative case the tuple $P.t_i$ is marked as deleted in all worlds that take values from t_C. For that, $t_C.(P.t_i.A)$ is assigned value $\perp$, and all other attributes $P.t_i.A'$ of C referring to the same tuple t_i of P are assigned value $\perp$ in t_C. This assures that if we later project away the attribute A of P, we do not erroneously "reintroduce" tuple $P.t_i$ into worlds that take values from t_C.

Example 4.1 Figure 8 shows the answers to $\sigma_{C=7}(R)$ and $\sigma_{B=1}(R)$. Note that the resulting WSDs should contain both the query answer P and the original relation R, but due to space limitations we only show the representation of P. One can observe that for both results in Figure 8 we obtain worlds of different sizes. For example the worlds that take values from the first tuple of the second component relation in Figure 8 (a) do not have a tuple t_1, while the worlds that take values from the second tuple of that component relation contain t_1. $\square$

Selection with condition $A\theta B$. The main added difficulty of selections with conditions $A\theta B$ as compared to selections with conditions $A\theta c$ is that it creates dependencies between two attributes of a tuple, which do not necessarily reside in the same component.

As the current decomposition may not capture exactly the combinations of values satisfying the join condition, components that have values for A and B of the same tuple are composed. After the composition phase, the selection algorithm follows the pattern of the selection with constant.

Example 4.2 Consider the query $\sigma_{A=B}(R)$, where R is represented by the 7-WSD of Figure 7. Figure 9 shows the query answer, which is a 4-WSD that represents five worlds, where one world has three tuples, three worlds have two tuples each, and one world has one tuple. $\square$

Product. The product $T := R \times S$ of two relations R and S, which have disjunct attribute sets and are represented by

R.t_1.A
1
2

×

R.t_1.B	R.t_1.C	R.t_2.B
1	0	3
2	7	4

×

R.t_2.A
4
5

×

R.t_2.C
0

×

R.t_3.A
6

×

R.t_3.B
6

×

R.t_3.C
7

Figure 7. 7-WSD representing a a set of 8 worlds.

P.t_1.A
1
2

×

P.t_1.B	P.t_1.C	P.t_2.B
$\perp$	$\perp$	3
2	7	4

×

P.t_2.A
4
5

×

P.t_2.C
$\perp$

×

P.t_3.A
6

×

P.t_3.B
6

×

P.t_3.C
7

(a) $P := \sigma_{C=7}(R)$ applied to the WSD of Figure 7.

P.t_1.A
1
2

×

P.t_1.B	P.t_1.C	P.t_2.B
1	0	$\perp$
$\perp$	$\perp$	$\perp$

×

P.t_2.A
4
5

×

P.t_2.C
0

×

P.t_3.A
6

×

P.t_3.B
$\perp$

×

P.t_3.C
7

(b) $P := \sigma_{B=1}(R)$ applied to the WSD of Figure 7.

Figure 8. Selections $P := \sigma_{C=7}(R)$ and $P := \sigma_{B=1}(R)$ with R from Figure 7.

P.t_1.A	P.t_1.B	P.t_1.C	P.t_2.A	P.t_2.B
1	1	0	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	4	4
2	2	7	4	4
2	2	7	$\perp$	$\perp$

×

P.t_2.C
0

×

P.t_3.A	P.t_3.B
6	6

×

P.t_3.C
7

Figure 9. $P = \sigma_{A=B}(R)$ with R from Figure 7.

R.t_1.A
1
2

×

R.t_1.B	R.t_2.A
3	5
4	6

×

R.t_2.B
7
8

×

S.t_1.C
a
b

×

S.t_1.D	S.t_2.C
c	e
d	f

×

S.t_2.D
g
h

(a) WSD of two relations R and S.

t_{11}.A	t_{12}.A
1	1
2	2

×

t_{11}.B	t_{12}.B	t_{21}.A	t_{22}.A
3	3	5	5
4	4	6	6

×

t_{21}.B	t_{22}.B
7	7
8	8

×

t_{11}.C	t_{21}.C
a	a
b	b

×

t_{11}.D	t_{21}.D	t_{12}.C	t_{22}.C
c	c	e	e
d	d	f	f

×

t_{12}.D	t_{22}.D
g	g
h	h

(b) WSD of their product $R \times S$.

Figure 10. The product operation $R \times S$.

a WSD requires that the product relation T extends a component C with $|S|_{max}$ (respectively $|R|_{max}$) copies of each column of C with values of R (respectively S). Additionally, the ith (jth) copy is named $T.t_{ij}.A$ if the original has name $R.t_i.A$ or $S.t_j.A$.

Example 4.3 Figure 10 (b) shows the WSD for the product of relations R and S represented by the WSD of Figure 10 (a). To save space, the relations R and S have been removed from Figure 10 (b), and attribute names do not show the relation name "T". □

Projection. A projection $P = \pi_U(R)$ on an attribute set U of a relation R represented by the WSD $\mathcal{C}$ is translated into (1) the extension of $\mathcal{C}$ with the copy P of R, and (2) projections on the components of $\mathcal{C}$, where all component attributes that do not refer to attributes of P in U are discarded. Before removing attributes, however, we need to propagate $\perp$-values, as discussed in the following example.

Example 4.4 Consider the 3-WSD of Figure 11 (a) representing a set of two worlds for R, where one world contains only the tuple t_1 and the other contains only the tuple t_2. Let P' represent the first two components of R, which contain all values for the attribute A in both tuples. The relation P' is not the answer to $\pi_A(R)$, because it encodes one world with *both* tuples, and the information from the third component of R that only one tuple appears in each world is lost. To compute the correct answer, we progressively (1) compose the components referring to the same tuple (in this case all three components), (2) propagate $\perp$-values within the same tuple, and (3) project away the irrelevant attributes. The correct answer P is given in Figure 11 (b).

Despite the component merging done by the projection, the size of the answer does not grow exponentially, because of attribute pruning and propagation of $\perp$-values. □

R.t_1.A
a

×

R.t_2.A
b

×

R.t_1.B	R.t_2.B
c	⊥
⊥	d

(a) WSD for R.

P.t_1.A	P.t_2.A
a	⊥
⊥	b

(b) WSD for P.

Figure 11. Projection $P := \pi_A(R)$.

Union. The algorithm for computing the union $T := R \cup S$ of two relations R and S works similarly to that for the product. Each component C containing values of R or S is extended such that in each world of C all values of R and S become also values of T.

Renaming. The operation $\delta_{A \to A'}(R)$ renames attribute A of relation R to A' as follows. For each tuple t of R, let C be the component that has the attribute $R.t.A$. Then we rename this attribute to $R.t.A'$ as $C := \delta_{R.t.A \to R.t.A'}(C)$.

Difference. To compute the difference operation $T := R - S$ we scan the components of the two relations R and S. For the worlds where a tuple t from R does not appear in S, t becomes a tuple of T; otherwise we place $\perp$-values to denote that t is not in these worlds of T.

Normalizing WSDs. The normalization of a WSD is the process of finding an equivalent WSD that takes the least space among all its equivalents. Examples of not normalized WSDs are non-maximal WSDs or WSDs defining invalid tuples (i.e., tuples that do not appear in any world). Note that removing invalid tuples and maximizing world-set decompositions can be performed in polynomial time [6].

Example 4.5 The WSD of Figure 8 (a) has only $\perp$-values for $P.t_2.C$. This means that the tuple t_2 of P is absent (or invalid) in all worlds and can be removed. The equivalent WSD of Figure 12 shows the result of this operation. Similar simplifications apply to the WSD of Figure 8 (b), where tuples t_2 and t_3 are invalid. □

P.t_1.A
1
2

×

P.t_1.B	P.t_1.C
⊥	⊥
2	7

×

P.t_3.A
6

×

P.t_3.B
6

×

P.t_3.C
7

Figure 12. Normalization of WSD of Figure 8 (a).

Example 4.6 The 4-WSD of Figure 9 admits the equivalent 5-WSD, where the third component is decomposed into two components. This non-maximality case cannot appear for UWSDTs, because all but the first component contain only one tuple and are stored in the template relation, where no component merging occurs. □

5 Experimental Evaluation

The literature knows a number of approaches to representing incomplete information databases, but little work has been done so far on expressive yet efficient representation systems. An ideal representation system would allow a large set of possible worlds to be managed using only a small overhead in storage space and query processing time when compared to a single world represented in a conventional way. In the previous sections we presented the first step towards this goal. This section reports on experiments with a large census database with noise represented as a UWSDT.

Setting. The experiments were conducted on a 3GHz/2GB Pentium machine running Linux 2.6.8 and PostgreSQL 8.0.

Datasets. The IPUMS 5% census data (Integrated Public Use Microdata Series, 1990) [18] used for the experiments is the publicly available 5% extract from the 1990 US census, consisting of 50 (exclusively) multiple-choice questions. It is a relation with 50 attributes and 12491667 tuples (approx. 12.5 million). The size of this relation stored in PostgreSQL is ca. 3 GB. We also used excerpts representing the first 0.1, 0.5, 1, 5, 7.5, and 10 million tuples.

Adding Incompleteness. We added incompleteness as follows. First, we generated a large set of possible worlds by introducing noise. After that, we cleaned the data by removing worlds inconsistent with respect to a given set of dependencies. Both steps are detailed next.

We introduced noise by replacing some values with or-sets[2]. We experimented with different noise densities: 0.005%, 0.01%, 0.05%, 0.1%. For example, in the 0.1% scenario one in 1000 fields is replaced by an or-set. The size of each or-set was randomly chosen in the range $[2, min(8, size)]$, where $size$ is the size of the domain of the respective attribute (with a measured average of 3.5 values per or-set). In one scenario we had far more than 2^{624449} worlds, where 624449 is the number of the introduced or-sets and 2 is the minimal size of each or-set (cf. Figure 13).

We then performed data cleaning using 12 equality generating dependencies, representing real-life constraints on the census data. Note that or-set relations are not expressive enough to represent the cleaned data with dependencies.

To remove inconsistent worlds with respect to given dependencies, we adapted the Chase technique [1] to the context of UWSDTs. We explain the Chase by an example. Consider the dependency WWII = 1 ⇒ MILITARY != 4 that requires people who participated in the second world war to have completed their military service. Assume now the dependency does not hold for a tuple t in some world and let C_1 and C_2 be the components defining t.WWII and t.MILITARY, respectively. First, the Chase computes a component C that defines both t.WWII and t.MILITARY. In case C_1 and C_2 are different, they are replaced by a new component $C = C_1 \times C_2$; otherwise, C is C_1. The Chase removes then from C all inconsistent worlds w, i.e., worlds

[2]We consider it infeasible both to iterate over all worlds in secondary storage, or to compute UWSDT decompositions by comparing the worlds.

	Density	0.005%	0.01%	0.05%	0.1%
Initial	#comp	31117	62331	312730	624449
After chase	#comp	30918	61791	309778	612956
	#comp>1	249	522	2843	10880
	$\|C\|$	108276	217013	1089359	2150935
	$\|R\|$	12.5M	12.5M	12.5M	12.5M
After Q_1	#comp	702	1354	7368	14244
	#comp>1	1	4	40	158
	$\|C\|$	1742	3625	19773	37870
	$\|R\|$	46600	46794	48465	50499
After Q_2	#comp	25	56	312	466
	#comp>1	0	1	8	9
	$\|C\|$	93	269	1682	2277
	$\|R\|$	82995	83052	83357	83610
After Q_3	#comp	38	76	370	742
	#comp>1	0	0	0	0
	$\|C\|$	89	202	1001	2009
	$\|R\|$	17912	17936	18161	18458
After Q_4	#comp	1574	3034	15776	30729
	#comp>1	11	28	127	557
	$\|C\|$	4689	9292	48183	94409
	$\|R\|$	402345	402524	404043	405869
After Q_5	#comp	3	10	53	93
	#comp>1	3	10	53	93
	$\|C\|$	1221	5263	33138	50780
	$\|R\|$	150604	173094	274116	393396
After Q_6	#comp	97	189	900	1888
	#comp>1	0	0	0	0
	$\|C\|$	516	1041	4993	10182
	$\|R\|$	229534	230113	234335	239488

Figure 13. UWSDTs characteristics for 12.5M tuples.

where w.WWII = 1 and w.MILITARY = 4. Repeating these steps iteratively for each dependency on a given UWSDT yields a UWSDT satisfying all dependencies.

$Q_1 := \sigma_{\text{YEARSCH}=17 \wedge \text{CITIZEN}=0}(R)$

$Q_2 := \pi_{\text{POWSTATE,CITIZEN,IMMIGR}}(\sigma_{\text{CITIZEN}<>0 \wedge \text{ENGLISH}>3}(R))$

$Q_3 := \pi_{\text{POWSTATE,MARITAL,FERTIL}}(\sigma_{\text{POWSTATE}=\text{POB}}(\sigma_{\text{FERTIL}>4 \wedge \text{MARITAL}=1}(R)))$

$Q_4 := \sigma_{\text{FERTIL}=1 \wedge (\text{RSPOUSE}=1 \vee \text{RSPOUSE}=2)}(R)$

$Q_5 := \delta_{\text{POWSTATE} \rightarrow P_1}(\sigma_{\text{POWSTATE}>50}(Q_2)) \bowtie_{P_1=P_2} \delta_{\text{POWSTATE} \rightarrow P_2}(\sigma_{\text{POWSTATE}>50}(Q_3))$

$Q_6 := \pi_{\text{POWSTATE,POB}}(\sigma_{\text{ENGLISH}=3}(R))$

Figure 14. Queries on IPUMS census data.

Figure 13 shows the effect of chasing our dependencies on the 12.5 million tuples and varying placeholder density. As a result of merging components, the number of components with more than one placeholder (#comp>1) grows linearly with the increase of placeholder density, reaching about 1.7% of the total number of components (#comp) in the 0.1% case. A linear increase is witnessed also by the chasing time when the number of tuples is also varied.

Queries. Six queries were chosen to show the behavior of relational operators combinations under varying selectivities (cf. Figure 14). Query Q_1 returns the entries of US citizens with PhD degree. The less selective query Q_2 returns the place of birth of US citizens born outside the US that do not speak English well. Query Q_3 retrieves the entries of widows that have more than three children and live in the state where they were born. The very unselective query Q_4 returns all married persons having no children. Query Q_5 uses query Q_2 and Q_3 to find all possible couples of widows with many children and foreigners with limited English language proficiency in US states with IPUMS index greater than 50 (i.e., eight 'states', e.g., Washington, Wisconsin, Abroad). Finally, query Q_6 retrieves the places of birth and work of persons speaking English well.

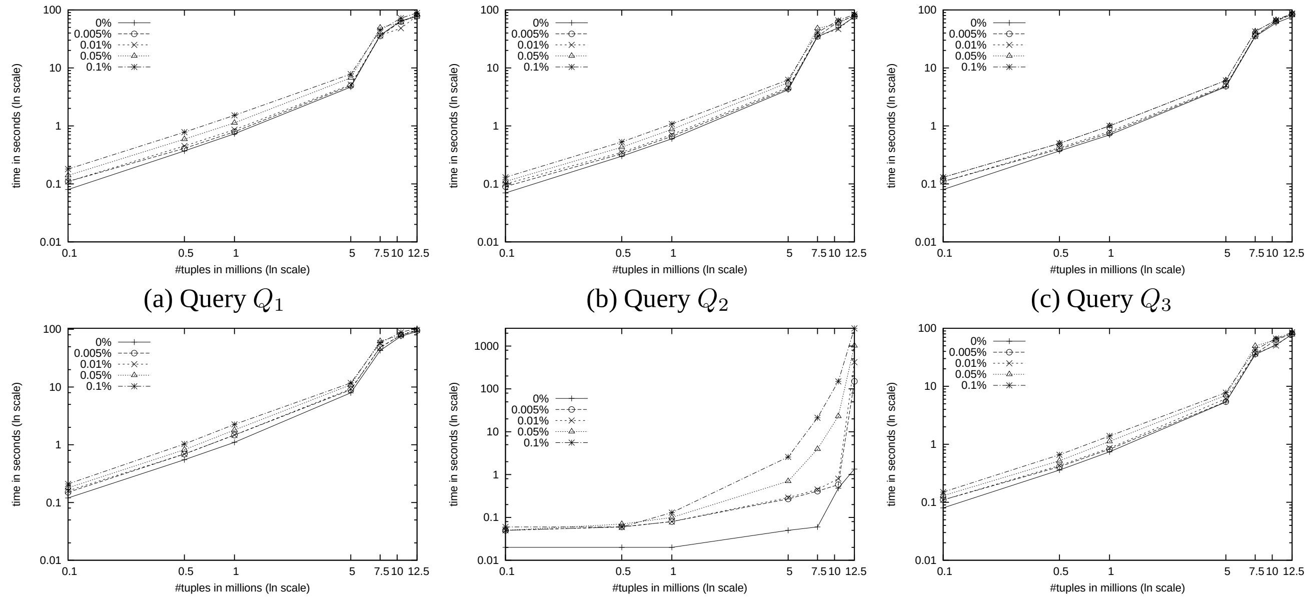
Figure 13 describes some characteristics of the answers to these queries when applied on the cleaned 12.5M tuples of IPUMS data: the total number of components (#comp) and of components with more than one placeholder (#comp>1), the size of the component relation C, and the size of the template relation R. One can observe that the number of components increases linearly with the placeholder density and that compared to chasing, query evaluation leads to a much smaller amount of component merging.

Figure 15 shows that all six queries admit efficient and scalable evaluation on UWSDTs of different sizes and placeholder densities. For accuracy, each query was run ten times, and the median time for computing and storing the answer is reported. The evaluation time for all queries but Q_5 on UWSDTs follows very closely the evaluation time in the one-world case. The one-world case corresponds to density 0% in our diagrams, i.e., when no placeholders are created in the template relation and consequently there are no components. In this case, the original queries (that is, not the rewritten ones) of Figure 14 were evaluated only on the (complete) template relation.

An interesting issue is that all diagrams of Figure 15 show a substantial increase in the query evaluation time for the 7.5M case. As the jump appears also in the one-world case, it suggests poor memory management of Postgres in the case of large tables. We verified this statement by splitting the 12.5M table into chunks smaller than 5M and running query Q_1 on those chunks to get partial answers. The final answer is represented then by the union of each UWSDT relation from these partial answers.

Although the evaluation of join conditions on UWSDTs can require theoretically exponential time (due to the composition of some components), our experiments suggest that they behave well in practical cases, as illustrated in Figures 15 (c) and (e) for queries Q_3 and Q_5 respectively. Note that the time reported for Q_5 does not include the time to evaluate its subqueries Q_2 and Q_3.

In summary, our experiments show that UWSDTs behave very well in practice. We found that the size of UWSDTs obtained as query answers remains close to that



(a) Query Q_1 (b) Query Q_2 (c) Query Q_3

(d) Query Q_4 (e) Query Q_5 (f) Query Q_6

Figure 15. The evaluation time for queries of Figure 14 on UWSDTs of various sizes and densities.

of one of their worlds. Furthermore, the processing time for queries on UWSDTs is comparable to processing one world. The explanation for this is that in practice there are rather few differences between the worlds. This keeps the mapping and component relations relatively small and the lion's share of the processing time is taken by the templates, whose sizes are about the same as of a single world.

Acknowledgments The authors were supported in part by DFG grant # KO 3491/1-1. The first author is supported by the International Max Planck Research School for Computer Science, Saarbrücken, Germany.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theor. Comput. Sci.*, 78(1):158–187, 1991.

[3] P. Andritsos, A. Fuxman, and R. J. Miller. Clean answers over dirty databases: A probabilistic approach. In *Proc. ICDE*, 2006.

[4] L. Antova, C. Koch, and D. Olteanu. 10^{10^6} worlds and beyond: Efficient representation and processing of incomplete information. Technical Report cs.DB/0606075, ACM CORR, 2006.

[5] L. Antova, C. Koch, and D. Olteanu. MayBMS: A possible worlds base management system. Web page. http://www.infosys.uni-sb.de/projects/maybms, 2006.

[6] L. Antova, C. Koch, and D. Olteanu. World-set decompositions: Expressiveness and efficient algorithms. In *Proc. ICDT*, 2007.

[7] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proc. PODS*, pages 68–79, 1999.

[8] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *Proc. VLDB*, 2006.

[9] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. "A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. In *Proc. SIGMOD*, 2005.

[10] A. Calí, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. PODS*, pages 260–271, 2003.

[11] R. Cheng, S. Singh, and S. Prabhakar. U-DBMS: A database system for managing constantly-evolving data. In *Proc. VLDB*, pages 1271–1274, 2005.

[12] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *Proc. VLDB*, pages 864–875, 2004.

[13] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. "AJAX: An Extensible Data Cleaning Tool". In *Proc. SIGMOD*, 2000.

[14] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of ACM*, 31:761–791, 1984.

[15] T. Imielinski, S. Naqvi, and K. Vadaparty. Incomplete objects — a data model for design and planning applications. In *Proc. SIGMOD*, pages 288–297, 1991.

[16] E. Rahm and H. H. Do. "Data Cleaning: Problems and Current Approaches". *IEEE Data Engineering Bulletin*, 2000.

[17] V. Raman and J. Hellerstein. "Potter's Wheel: An Interactive Data Cleaning System". In *Proc. VLDB*, 2001.

[18] S. Ruggles, M. Sobek, T. Alexander, C. A. Fitch, R. Goeken, P. K. Hall, M. King, and C. Ronnander. Integrated public use microdata series: V3.0, 2004. http://www.ipums.org.

[19] A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *Proc. ICDE*, 2006.