

Efficient Representation and Processing of Incomplete Information

Lyublena Antova

February 2006

Master Thesis in Computer Science
Saarland University

Supervisors: Prof. Dr. Christoph Koch
Dr. Dan Olteanu

I hereby declare that this thesis is entirely my own work except where otherwise indicated.

I have used only the resources given in the list of references.

Lyublena Antova
Saarbrücken, February 2006

Acknowledgements

I would like to thank my supervisor Prof. Christoph Koch for his inspiring courses that motivated me to work in this field. I am grateful for the many helpful discussions with him and Dr. Dan Olteanu during the writing of this thesis.

I would also like to thank the IMPRS team headed by Kerstin Meyer-Ross for providing a friendly environment conducive to research.

Last but not least I would like to thank my family and friends.

Abstract

Database systems often have to deal with incomplete information as the world they model is not always complete. This is a frequent case in data integration applications, scientific databases, or in scenarios where information is manually entered and contains errors or ambiguity.

In the last couple of decades different formalisms have been proposed for representing incomplete information. These include, among other, the so-called relations with or-sets, tables with variables (v-tables) and conditional tables (c-tables).

However, none of the current approaches for representing incomplete information has satisfied the requirements for a powerful and efficient data management system, which is the reason why none has found application in practice. All models generally suffer from at least one of two weaknesses. Either they are not strong enough for representing results of simple queries, as is the case for v-tables and relations with or-sets, or the handling and processing of the data, e.g. for query evaluation, is intractable (as is the case for c-tables).

In this thesis, we present a decomposition-based approach to addressing the problem of incompletely specified databases. We introduce *world-set decompositions (WSDs)*, a space-efficient formalism for representing any finite set of possible worlds over relational databases. WSDs are therefore a strong representation system for any relational query language.

We study the problems of evaluating relational algebra queries on WSDs. For each relational algebra operation we present an algorithm operating on WSDs.

We also address the problem of data cleaning in the context of world-set decompositions. We present a modified version of the existing Chase algorithm, which we use to remove inconsistent worlds in an incompletely specified database.

We evaluate our techniques in a large census data scenario with data originating from the 1990 USA census and we show that data processing on WSDs is both scalable and efficient.

Contents

1	Introduction	1
2	Related Work	9
3	Preliminaries	17
4	World-Set Decompositions	19
4.1	The Basic Notion	19
4.2	Adding Template Relations	21
4.3	Uniform World-Set Decompositions	22
5	Queries on Decompositions	25
5.1	Selection with condition $A\theta c$	26
5.2	Selection with condition $A\theta B$	29
5.3	Projection	31
5.4	Product	34
5.5	Union	35
5.6	Renaming	35
5.7	Difference	36
5.8	Discussion	36
5.9	Simplifying WSDs	38
6	Efficient Query Evaluation on UWSDTs	41
7	Chasing Dependencies	45
8	Experimental Evaluation	51
9	Conclusions and Future Work	59
	Schema and codes for IPUMS data	61

List of Figures

1.1	Two completed survey forms.	3
1.2	World-set relation for the remaining 24 worlds after excluding the ones with duplicated social security numbers.	4
1.3	WSD of the world-set relation of Figure 1.2.	4
2.1	Truth tables for the three-valued logic.	9
2.2	Naive table.	10
2.3	Conditional tables.	10
2.4	Representing incomplete XML documents.	11
2.5	Defining transformations in AJAX.	13
4.1	A WSD with a template.	21
4.2	A uniform WSD for our running example.	23
4.3	A UWSDT corresponding to the WSDT of Figure 4.1.	23
5.1	World-set and its decomposition	27
5.2	Selections $P := \sigma_{C=7}(R)$ and $P := \sigma_{B=1}(R)$ with R from Figure 5.1 (b).	28
5.3	$P = \sigma_{A=B}(R)$ with R from Figure 5.1 (b).	30
5.4	$P := \pi_B(R)$ with R from Figure 5.1 (b).	31
5.5	Evaluating projection when \perp -values are involved.	32
5.6	The product operation $R \times S$	34
5.7	The union $T = R \cup S$ for the relations in Figure 5.6 (a).	35
5.8	WSD of two relations R and S	36
5.9	The difference operation $R - S$	37
5.10	WSD Simplification Rules.	38
5.11	Simplification of WSD of Figure 5.2 (a).	39
6.1	Composing components in WSDs.	42
6.2	Composing components in UWSDs.	43
7.1	Result of chasing $S = 785 \Rightarrow M = 1$ on the WSD in Figure 1.3.	46
7.2	Impact of order on chasing	49

8.1	Dependencies for cleaning census data	52
8.2	Time for chasing the dependencies of Figure 8.1 on UWSDTs of various sizes and densities.	53
8.3	UWSDTs characteristics before chasing and after chasing and querying for 12.5M tuples	54
8.4	Queries on IPUMS census data	55
8.5	Evaluation time for Query Q_1	55
8.6	Evaluation time for Query Q_2	56
8.7	Evaluation time for Query Q_3	56
8.8	Evaluation time for Query Q_4	57
8.9	Evaluation time for Query Q_5	57
8.10	Evaluation time for Query Q_6	58

List of Algorithms

1	Selection with constant	27
2	Propagation of \perp -values	28
3	Selection with a join condition	31
4	Projection	33
5	Product	34
6	Union	35
7	Rename	36
8	Selection with constant on UWSDTs.	42
9	Chase	48

Chapter 1

Introduction

Databases with incompletely specified information appear often in practice. Classical examples of incompleteness can be found in data integration applications, linguistic data collections, or whenever information is manually entered and is therefore prone to inaccuracy or partiality. Although such examples can be considered the rule rather than just an exception, there has been little research towards powerful yet scalable systems for representing incomplete information.

An important property of *representation systems* is the ability to represent the answer of a query in a given language using the same formalism in which the original database is defined. A representation system which has this property is called a *strong representation system* with respect to a query language [17].

Current techniques for representing incomplete information can be classified into two groups. The first group includes representation systems such as *v-tables* [17] and *relations with or-sets* [18] which are not strong for relational algebra or even important fragments such as select-project-join queries. 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* (tables with conditions) [17] belong to the second group of formalisms. They extend *v-tables* by adding logical conditions on the variables, thus constraining the possible values. Although the *c-tables* are a strong representation system, they have not found application in practice. The main reason for this is probably that they are awkward to handle; storing *c-tables* directly is rather inefficient, and it is known that already the *data complexity* [27], i.e. the complexity of the query evaluation problem under the assumption

that the size of queries is fixed, of relational algebra over data represented using *c*-tables is NP-hard [14]. Of course, this is in stark contrast to the data complexity of relational algebra over classical relations (i.e., a completely specified relational database), which is not only in polynomial time but even in logarithmic space and highly parallelizable (cf. e.g. [1]).

As a motivation, let us consider the following example. Figure 1.1 shows two manually completed forms that may originate from a census or some other kind of survey and in which portions allow for more than one interpretation. For simplicity we assume that social security numbers consist only of three digits. For instance, Smith’s social security number can be read either as “185” or as “785”. Nevertheless it is still possible to represent the available information using relations 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 $2*2*2*4 = 32$ 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 the use of *data cleaning* procedures by which certain unlikely or invalid worlds can be excluded. However, the results of both types of operation turn out not to be representable by or-set relations in general. Consider for example the following query: ”Find all pairs of people that have a different marital status”. Each tuple in the result contains two marital status fields, which cannot have the same value at the same time. This renders some combinations of values invalid, and the query result cannot be represented as an or-set relation.

An example integrity constraint for data cleaning that makes it impossible to store the data as an or-set relation is the requirement that all social security numbers should be unique. For our example database, this rule will eliminate 8 of the 32 worlds, namely the ones in which both tuples have the value 185 as social security number.

It is impossible to represent the 24 worlds that remain after applying the data cleaning rule from above using or-set relations. What we could do is store each possible world explicitly. One way to do this is using a table called a *world-set relation* of a given set of possible worlds. Each tuple in this table represents one world and is the concatenation of all tuples in that world (see Figure 1.2).

Social Security Number:	<u>785</u>
Name:	<u>Smith</u>
Marital Status:	(1) single <input checked="" type="checkbox"/> (2) married <input checked="" type="checkbox"/> (3) divorced <input type="checkbox"/> (4) widowed <input type="checkbox"/>

Social Security Number:	<u>185</u>
Name:	<u>Brown</u>
Marital Status:	(1) single <input type="checkbox"/> (2) married <input type="checkbox"/> (3) divorced <input type="checkbox"/> (4) widowed <input type="checkbox"/>

Figure 1.1: Two completed survey forms.

The most striking problem of world-set relations is their size. If we conduct a survey of 250 questions on a population of $3 * 10^8$ and we assume that every 10000th answer can be read in just two different ways, we will get about 2^{10^6} possible worlds. Each such world is a substantial table of 250 columns and $3 * 10^8$ rows. We cannot store all these worlds explicitly in a world-set relation. Data cleaning will often eliminate only some of these worlds, so a DBMS should be able to manage those that remain.

In this thesis, we aim at dealing with this complexity. Our approach is based on 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 \times of relational algebra) is again the world-set relation.

Example 1.0.1 The world-set represented by our initial or-set relation can also be represented by the product

$$\begin{array}{|c|} \hline t_1.S \\ \hline 185 \\ 785 \\ \hline \end{array} \times \begin{array}{|c|} \hline t_1.N \\ \hline Smith \\ \hline \end{array} \times \begin{array}{|c|} \hline t_1.M \\ \hline 1 \\ 2 \\ \hline \end{array} \times \begin{array}{|c|} \hline t_2.S \\ \hline 185 \\ 186 \\ \hline \end{array} \times \begin{array}{|c|} \hline t_2.N \\ \hline Brown \\ \hline \end{array} \times \begin{array}{|c|} \hline t_2.M \\ \hline 1 \\ 2 \\ 3 \\ 4 \\ \hline \end{array}$$

Example 1.0.2 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 1.3.

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

Figure 1.2: World-set relation for the remaining 24 worlds after excluding the ones with duplicated social security numbers.

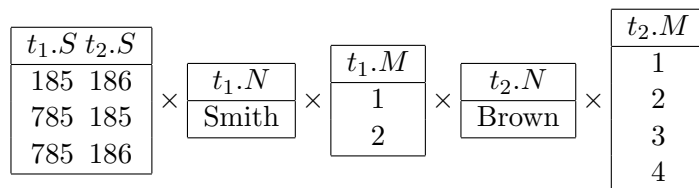


Figure 1.3: WSD of the world-set relation of Figure 1.2.

One can observe that the result of this product is exactly the world-set relation in Figure 1.2. 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 separate components. \square

Using WSDs we can represent sets of worlds that cannot be expressed with or-set relations; at the same time the decomposition is just slightly larger than the original or-set relation for those cases where world-sets are actually expressible as or-set relations and can be compared.

A particular advantage of this approach is that we do not need to assign “global” unique identifiers (IDs) to the individual worlds. This is important, since the size of the IDs can exceed the size of the decomposition itself, thus making it difficult or even impossible to represent the world-sets in a space-efficient way.¹

The technical contributions of this thesis are as follows.

- We formally introduce WSDs 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. WSDs can represent any finite set of possible worlds over relational databases and are therefore a strong representation system for *any relational query language*.
- In practice, it is often the case that corresponding 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 present a refinement of WSDs, *WSDTs*, that stores information that is the same in all possible worlds once and for all in so-called *template relations*. WSDTs thus combine the advantages of WSDs and v-tables.

- WSDs and WSDTs as representation systems have a practical problem, namely that the DBMS that manages WSDs has to support relations of arbitrary arity. (Indeed, the schemata of the component relations of a decomposition depend on the data.) Unfortunately, database systems (e.g. PostgreSQL) in practice often do not support relations beyond a fixed arity.

¹If any world-set over a given schema and a fixed active domain is permitted, one can verify that global world-ids cannot be smaller than the largest possible world over the schema and the active domain.

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 first discuss the query evaluation problem in the context of the much more graphic WSDs.

We also develop a number of optimizations and techniques for simplifying the data representations obtained by queries to support scalable query processing even on very large world-sets.

- We initiate a study of the data cleaning problem in the context of UWSDTs. We focus on two kinds of dependencies, functional dependencies and a class of (in)equality-generating dependencies, and adapt the *Chase procedure* (cf. [22, 4, 15]) for incomplete information to the framework of UWSDTs.
- We briefly describe a prototype implementation on top of the PostgreSQL RDBMS that 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 yield highly scalable techniques for managing incomplete information. We found that the size of UWSDTs obtained as query answers or as result of chasing remains close to that of a single world. Furthermore, our representation system has the property that the processing time for queries is also comparable to processing just a single world and thus a classical relational database.

A fundamental assumption of this work is that one wants to store and manage *sets* of possible worlds. We believe that this is justified by previous work on strong representation systems, starting with Imielinski and Lipski [17], and by current application requirements. Data cleaning is often an incremental process that requires to store large intermediate results, world-sets, in databases. Our approach can deal with data in which not all uncertainties could be resolved. Such databases are still valuable. It should be possible to do data transformation queries that preserve as much information as possible, thus necessarily mapping from sets of possible worlds to sets of possible worlds. A different approach is followed in work on finding *certain answers* of queries on incomplete-information databases (see e.g. [5]).

Regarding data cleaning, we demonstrate that it is well feasible on WSDs. In our experiments, data cleaning provides us some interesting world-sets to run queries on. Further work will be required to close the gap between our simple dependency-theoretic framework and the state-of-the art in data cleaning [23, 13, 24, 8, 7], but we hope that the present thesis renders it plausible to the reader that the framework of WSDs is relevant for data cleaning as well.

The structure of this thesis is as follows. We start by reviewing related work in the field of incomplete information and data cleaning (Chapter 2) and introducing some required notation in Chapter 3. In Chapter 4, we formally define the WSDTs and describe their extension to UWSDTs for the practical use. Next we provide algorithms for evaluating queries in our framework. We start by showing query processing on WSDs in Chapter 5 and discuss the extension to UWSDTs in Chapter 6. Chapter 7 addresses the data cleaning problem on WSDTs by providing a modification of the Chase algorithm. Finally, Chapter 8 presents our experimental evaluation of the approach.

Chapter 2

Related Work

Incomplete Information

There has been interest in representing and querying incomplete information since Codd introduced the relational data model in 1970 [9]. To deal with missing values (denoted by '@'), he proposes an extension to the relational algebra [10, 11]. It is based on three-valued logic (see Figure 2.1), and the so-called *null substitution principle*. As was observed, however, this approach leads to erroneous answers to tautological queries, such as "select * from R where $a < 5$ or $a \geq 5$ " when the a column contains null values.

A	B	A and B	A	B	A or B	A	not(A)
f	f	f	f	f	f	f	t
t	f	f	t	f	t	t	f
f	@	f	f	@	@	@	@
t	@	@	t	@	t	@	@

Figure 2.1: Truth tables for the three-valued logic.

The seminal study on incomplete information in relational databases was conducted by Imielinski and Lipski in the early 1980s [17]. They define the notion of strong representation systems and study in detail the properties of three models for representing incomplete information - Codd tables, *v*-tables and *c*-tables. Codd tables encode missing information with null values. They are shown to support PS-queries, but not PSU- and PJ-queries. In *v*-tables null values are replaced by variables. *Naive tables* is a term used for *v*-tables that are allowed to contain a variable more than once. This makes it possible to specify join-conditions over the missing information. For example in the naive table of Figure 2.2 the use of the same variable in columns A and C in

the first tuple implies that the values are equal. Imielinski and Lipski show that naive tables are a strong representation system for positive relational algebra (SPJU-queries).

R	A	B	C
	x	5	x
	y	z	8

Figure 2.2: Naive table.

Conditional tables (c-tables) extend v-tables by adding explicit constraints on the variables. Consider for example the c-table in Figure 2.3 (a). The table contains an additional column *con*, which specifies a condition over the variables in the tuple. Thus in the first tuple x can take a value 5 or 6, and in the second tuple the value of y should be different from the one for z and in addition y should not be 7. Queries over a c-table may require enforcing additional constraints on the values, as for example the result of the selection $\sigma_{A < 6}(R)$ in Figure 2.3 (b). Imielinski and Lipski prove that c-tables are a strong representation system for relational algebra. Later Grahne shows that c-tables form a strong representation also for query languages with fixpoint on positive queries [14].

R	A	B	con	$\sigma_{A < 6}(R)$	A	B	con
	x	5	$x = 5 \vee x = 6$		x	5	$x = 5$
	y	z	$y \neq z \wedge y \neq 7$		y	z	$y \neq z \wedge y \neq 7 \wedge y < 6$
	(a) Example c-table				(b) Queries on c-tables		

Figure 2.3: Conditional tables.

Incomplete information has also been studied in the context of XML. Abiteboul et al. propose the so-called *incomplete trees* [3] by adapting the idea of c-tables to the tree model. They consider a scenario where information is gradually collected from the Web using answers from queries, and is stored in a centralized location. Initially, what is known about the document is its structure defined by a *tree type*. For example the tree type in Figure 2.4 (a) defines a valid XML document about university courses, where the structure is specified in terms of children relations and multiplicity constraints. Figure 2.4 (b) shows a prefix-selection query that finds all courses with more than five credit points. The database is augmented with the result of this operation (see Figure 2.4 (c)) and now contains information about the Logic

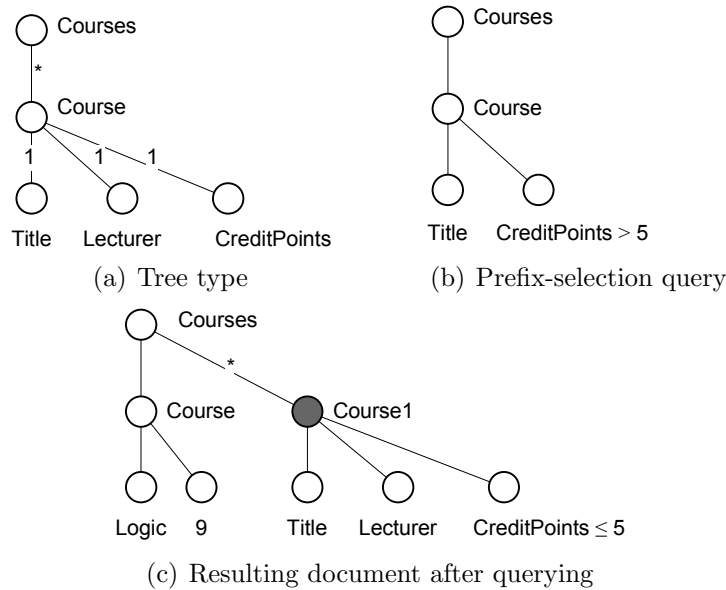


Figure 2.4: Representing incomplete XML documents.

course returned as an answer to the query. What is also known is that all other courses give at most five credit points. The latter is represented using *conditional tree types*, which specify a tree structure and associate conditions to nodes. In our example the missing information is depicted as the gray node labeled `Course1`. Abiteboul et al. show that incomplete trees form a strong representation system for prefix-selection queries. Some shortcomings of the model include a possible exponential blowup of the size of the incomplete trees, the need for persistent node ids and the lack of order among the nodes of the tree.

There are further applications where incomplete information may appear. One example is the theory of *views* in databases, cf. [19, 6]. Since views are often designed to hide information, they are incomplete with respect to the whole database. This causes difficulties when trying to update views, as only part of the information can be provided by the user.

The well founded semantics of Datalog with negation (cf. [1]) can also be viewed as a source of incompleteness. It uses a three-value logic to express that a certain fact is true or not; this may also be unknown.

The complexity of querying incompletely specified databases has been investigated in [2]. Abiteboul, Kanellakis and Grahne show that the data complexity of the q -membership problem, that is, deciding whether a given world is in the result of a query, is NP-complete when data is represented as

c-tables.

Data Cleaning

Data cleaning has been studied mostly in the context of data integration and warehousing, where information comes from different sources and is erroneous or inconsistent. Data cleaning is usually a semi-automatic process and most techniques treat only isolated aspects of the problem without providing an integral solution.

An important subproblem in data integration and cleaning is the so called Merge/Purge problem. It consists of *linking* records, that is, detecting equivalent items and eliminating duplicates. This proves to be a non-trivial task, as record identifiers may be unavailable, they may have different formatting in each dataset, or simply be erroneous.

The sorted-neighborhood method proposed by Hernandez and Stolfo [16] discovers equivalent records in a collection of merged datasets. The main idea is to sort the collection and inspect records using a sliding window approach, with the hope that equivalent records will appear close to each other in the sorted list. To perform the sorting a key for each record is computed, where the key may involve different attributes or parts of attributes. The test for equivalence includes computing a distance function measuring how close two records are. This may for example be the *edit distance*, which gives the minimum number of character edit operations (delete, insert and substitute) required to transform one of the tuples into the other. Other distance functions include *phonetic distance* or *typewriter distance*. If two records are close enough (as defined by a suitable threshold), they are considered to be related to the same entity. Since using a single key is unlikely to detect all matching records, the sorted-neighborhood method is further improved to a *multi-pass* version. The algorithm is executed multiple times over the dataset, each time using a different key for the sorting. The final set of groups of equivalent records is computed as a transitive closure over the results of each independent run of the algorithm.

The AJAX system developed at INRIA [13] allows users to define their own cleaning program. It provides a declarative and extensible language based on the syntax of SQL. The cleaning process is modeled as a directed graph of atomic transformations. The AJAX framework supports five types of atomic transformations. The *mapping* operation transforms a tuple in a certain way in order to standardize data formats. *Matching* finds pairs of similar records that refer to the same entity, where the test for similarity is performed via user-defined criteria. The *clustering* transformation groups

```
CREATE MATCHING M1
FROM Products p1, Products p2
LET similarity = sim(p1.name, p2.name)
WHERE p1.id < p2.id
AND similarity > 0.5
{SELECT p1.id AS id1, p2.id AS id2, similarity AS similarity
KEY id1, id2}

CREATE CLUSTER C1
FROM M1
BY transitiveClosure
```

Figure 2.5: Defining transformations in AJAX.

tuples by similarity, for example by computing the transitive closure over all pairs of similar records; the clusters can be then collapsed to a single tuple using the *merging* operation. The last atomic transformation is *SQL View*, which corresponds to an SQL join or union.

Figure 2.5 shows an example data cleaning program for AJAX. Given a table of products, it first finds all pairs of records having similar names, as computed by the *sim* function. The result is a data flow (specified in the curly braces), containing the ids of the matched tuples and their similarity measure. After that the program clusters the tuples by similarity using transitive closure.

AJAX supports a mechanism that allows users to interact with the system during the cleaning process. Whenever an exceptional situation arises when executing a transformation, for example a tuple violates an integrity constraint, AJAX marks the tuples that caused the exception together with an error report and presents them to the user at the end of the computation. The user can then decide on what to change in the specification of the cleaning program, or can ask for additional explanation of the cleaning process.

Potter’s Wheel by Raman and Hellerstein [24] adds interactivity to the cleaning process. The target is to discover and fix data items not matching a predefined structure. This may involve splitting or merging columns, applying a function to the values in a given column or demoting table and column names into column values. Transformations are defined incrementally by means of a graphical interface. When users observe incorrect data, what they need to do is specify the desired result on the example values and the system tries to infer a suitable transformation automatically. At the same time additional algorithms for detecting data anomalies run in the

background. In this way long waiting times caused by batch execution are avoided.

A recent work deals with finding low-cost repairs for databases violating a set of integrity constraints [7]. Bohannon et al. prove that when both functional and inclusion dependencies are involved and repairs are based on *value modification* (as opposed to tuple *insertion* and *deletion*), finding an optimal solution is in general NP-complete in the size of the data. Instead, they propose a heuristic based on two factors: accuracy and similarity. The *accuracy* captures the user's confidence in the correctness of the data and is assumed to be given as input. Repair cost is computed in terms of *similarity* of the original and the repaired data. Bohannon et al. use *equivalence classes* to group fields of the database that should have the same value. For example for a functional dependency $A_1, \dots, A_n \rightarrow A_0$ if a pair of tuples s and t matches on the attributes on the left-hand side of the dependency, then (s, A_0) and (t, A_0) are put in the same equivalence class. In this way the actual assigning of values is postponed.

Chaudhuri et al. study the problem of online data cleaning [8], where a data warehouse decides to accept or reject an incoming tuple, which possibly contains errors. The decision is based on whether the input tuple matches some acceptable tuple in a so called *reference table*. The reference table contains tuples that are known to be clean, for example valid names and addresses obtained from a postal department. The matching is performed by a *fuzzy match operation*, which returns with high probability the closest matching reference tuple. To measure the closeness of two tuples, the authors propose an extension of the edit distance similarity function. The input string is viewed as a series of tokens, where each token is assigned a weight based on the *inverse document frequency*, that is, the more often a token appears, the less important it is considered to be. Token importance is reflected in the similarity function by giving tuples matching on more important tokens higher similarity value. In addition the authors propose an indexing structure, the so-called *error tolerant index relation*, which speeds up the retrieval of small set of candidate matches at run time.

The *Chase* technique was first introduced in [4, 22] to test implications of data dependencies. It was shown that the Chase for functional and join dependencies possesses the Church-Rosser property, that is, it terminates and the resulting tableau is unique. Grahne uses the Chase to enforce a set of dependencies on c-tables [15].

An assumption in the data cleaning process is that if a constraint is not satisfied, the data is erroneous. However, this is not always true as the database might contain exceptions to the rule. In this case cleaning is undesirable as it might lead to loss of information. A different strategy for

managing inconsistent databases is to keep the data in its original state and guarantee consistency only when answering queries. This idea is based on the notion of *repair* - a clean version of the database. A *consistent query answer* consists of tuples that are answer to the query in every repair.

The problem of consistent query answers in inconsistent databases has been studied by Arenas, Bertossi and Chomicki [5], who propose a method for first-order query rewriting. They also study the data complexity of consistent query answers for different classes of queries and dependencies.

The ConQuer system built by Fuxman, Fazil and Miller [12] allows users to specify a set of key constraints together with their query. The system then rewrites the query into SQL and returns the set of consistent answers.

Chapter 3

Preliminaries

We use the named perspective of the relational model (cf. e.g. [1]). A *relation scheme* is of the form $R[U]$, where U is a set of attribute names. The arity $|U|$ of R will be denoted by $ar(R)$. A *relational schema* Σ is a set of relation schemes.

Let \mathbf{D} be a finite set of domain elements. A *relation* over scheme $R[A_1, \dots, A_k]$ is a set of tuples $(A_1 : a_1, \dots, A_k : a_k)$ where $a_1, \dots, a_k \in \mathbf{D}$. A *relational database* \mathcal{A} over schema Σ is a set of relations R^A , one for each relation scheme $R[U]$ from Σ . Sometimes, when no confusion of database may occur, we will use R rather than R^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.

We will use the named perspective of relational algebra, with the operations selection σ , projection π , product \times , union \cup , difference $-$, and attribute renaming δ (cf. [1]).

Let R be a relation with schema $R[U]$. Then a disjoint m -partition $\{U_1, \dots, U_m\}$ of U is called a (*product*) *m -decomposition* of R if and only if

$$\pi_{U_1}(R) \times \dots \times \pi_{U_m}(R) = R$$

An m -decomposition is called *maximal* (*ly decomposed*) if no n -decomposition of R exists with $n > m$.

Proposition 3.0.3 *For each relation a maximal decomposition exists and is unique.*

Proof. Existence is clear because a world-set relation is also a 1-WSD. Uniqueness is shown by contradiction: Given relation R of schema $R[U]$, let us assume that there are two different maximal m -decompositions

$\{U_1, \dots, U_m\}$ and $\{V_1, \dots, V_m\}$ of R . Since the two decompositions are different, there are two sets U_i, V_j such that $U_i \neq V_j$ and $U_i \cap V_j \neq \emptyset$. But then, as of course $R = \pi_{U-V_j}(R) \times \pi_{V_j}(R)$, we have

$$\pi_{U_i}(R) = \pi_{U_i}(\pi_{U-V_j}(R) \times \pi_{V_j}(R)) = \pi_{U_i-V_j}(R) \times \pi_{U_i \cap V_j}(R).$$

It follows that

$$\{U_1, \dots, U_{i-1}, U_i - V_j, U_i \cap V_j, U_{i+1}, \dots, U_m\}$$

is an $(m+1)$ -decomposition of R , and m -decompositions cannot be maximal. Contradiction. \square

A set of *possible worlds* (or *world-set*) over schema Σ is a set of databases over schema Σ .

Let \mathbf{R} be a set of structures, rep be a function that maps to world-sets of the same schema. Then rep is a *strong representation system* for a query language if, for each query Q of that language and each $\mathcal{R} \in \mathbf{R}$ such that Q is applicable to the worlds in $rep(\mathcal{R})$, there is a structure $\mathcal{R}' \in \mathbf{R}$ such that

$$rep(\mathcal{R}') = \{Q(\mathcal{A}) \mid \mathcal{A} \in rep(\mathcal{R})\}.$$

Obviously,

Lemma 3.0.4 *If rep is a function whose image is the set of all finite world-sets, then rep is a strong representation system for any relational query language.*

For the remainder of the thesis, we make the following

Proviso 3.0.5 *The arity of all our database relations is at least one. The projection operation does not project to the empty set of attributes.*

Chapter 4

World-Set Decompositions

4.1 The Basic Notion

In order to use classical database techniques for storing and querying 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 Σ . For each $R \in \Sigma$, let $|R|_{\max} = \max\{|R^{\mathcal{A}}| : \mathcal{A} \in \mathbf{A}\}$ denote the maximum cardinality of R in any world of \mathbf{A} . Given a world \mathcal{A} with $R^{\mathcal{A}} = \{t_1, \dots, t_{|R^{\mathcal{A}}|}\}$, let $t_{R^{\mathcal{A}}}$ be the tuple obtained as the concatenation (denoted \circ) of the tuples of $R^{\mathcal{A}}$ padded with a special null value $\perp \notin \mathbf{D}$ up to arity $ar(R) \cdot |R|_{\max}$,

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

Then tuple $t_{\mathcal{A}} := t_{R_1^{\mathcal{A}}} \circ \dots \circ t_{R_k^{\mathcal{A}}}$ for $\Sigma = \{R_1, \dots, R_k\}$ encodes all the information in world \mathcal{A} . Now, by the *world-set relation* $W(\mathbf{A})$ of world-set \mathbf{A} , we denote the relation $\{t_{\mathcal{A}} \mid \mathcal{A} \in \mathbf{A}\}$. Relation $W(\mathbf{A})$ has schema (attributes)

$$\{R.t_i.A_j \mid R[U] \in \Sigma, 1 \leq i \leq |R|_{\max}, A_j \in U\}.$$

Given a world-set relation W , let $rep_{WS}(W)$ denote the represented world-set \mathbf{A} , i.e. the world-set s.t. $W = W(\mathbf{A})$. Given the above definition that turned every world in a tuple in a canonical form, computing $rep_{WS}(W)$ 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 $t_{R^{\mathcal{A}}} = (a_1, \dots, a_{ar(R) \cdot |R|_{\max}})$ to $R^{\mathcal{A}}$ as

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

(So one can think of \perp as a deletion marker for tuples.)

Observe that although world-set relations are not unique as we have left the ordering in which the tuples of a given world are concatenated open, all world-set relations of a world-set \mathbf{A} are equally good for our purposes because rep_{WS} maps them invariantly back to \mathbf{A} .

Definition 4.1.1 Let \mathbf{A} be a world-set. Then a *world-set m -decomposition* (*m -WSD*) of \mathbf{A} is a tuple $\{C_1, \dots, C_m\}$ such that

$$C_1 \times \dots \times C_m = W(\mathbf{A})$$

that is, the schemas of $\{C_1, \dots, C_m\}$ constitute an m -decomposition of the world-set relation $W(\mathbf{A})$. The world-set represented by m -WSD $\{C_1, \dots, C_m\}$, subsequently called $rep(\{C_1, \dots, C_m\})$, is $rep_{WS}(C_1 \times \dots \times C_m)$.

Clearly, if we define maximality of world-set m -decompositions in analogy to that of m -decompositions (i.e., w.r.t. the components of a world-set m -decomposition), there is, given a world-set 1-decomposition, a unique maximally decomposed version of it.¹

Remark 4.1.2 There is a fairly large literature on the universal relation assumption and relational decomposition, particularly on lossless join decomposition, cf. e.g. [26, 1]; however, previous work assumes that decompositions are defined intensionally using dependencies. Decompositions of extensionally given relations are less natural in the classical context because such decompositions may break on updates.

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 Chapter 1. Further examples can be found in Chapter 5. 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 4.1.3 *Any finite set of possible worlds can be represented as a world-set relation and as a 1-WSD.*

Corollary 4.1.4 (Lemma 3.0.4) *WSDs are a strong representation system for any relational query language.*

¹Note that this is only true for the WSDTs defined below if the template relation R^0 is fixed.

As demonstrated in Chapter 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*.

4.2 Adding Template Relations

We now present our refinement of WSDs that uses *template relations* to store information that is the same in all possible worlds. These template relations place null values “?” in fields at which different worlds have different values.

We will assume that tuples t have unique ids \hat{t} . Let

$$\Sigma = \{R_1, \dots, R_k\}$$

be a schema and \mathbf{A} a finite set of possible worlds over Σ . Then, a database

$$(R_1^0, \dots, R_k^0, C_1, \dots, C_m)$$

is called an *m-WSD with template relations (m-WSDT)* of \mathbf{A} iff there is a WSD $(C_1, \dots, C_m, D_1, \dots, D_n)$ of \mathbf{A} such that $|D_i| = 1$ for all i and if relation D_i has attribute $R_j.\hat{t}.A$ and value v in its unique $R_j.\hat{t}.A$ -field, then R_j^0 has a tuple with id \hat{t} whose A -field has value v .

Example 4.2.1 Consider again the set of 24 worlds of the running example of Chapter 1. Let us change that example a bit to reduce the number of partially specified fields. If we assume that the marital status in t_2 has a value 3, we obtain a set of six worlds that can be represented by the 2-WSDT of Figure 4.1.

R^0	S	N	M		$R.t_1.S$	$R.t_2.S$		$R.t_1.M$
t_1	?	Smith	?	×	185	186		1
t_2	?	Brown	3		785	185		2
					785	186		

Figure 4.1: A WSD with a template.

Of course WSDTs again can represent any finite world-set and are thus a strong representation system for relation query languages.

4.3 Uniform World-Set Decompositions

Database systems in practice often do not support relations of arbitrary arity. For that reason we introduce a modified representation of WSDs called *uniform WSDs* next. We use the fixed schema consisting of the three relation schemes

$$F[FID, CID], W[CID, LWID], C[FID, LWID, VAL],$$

where FID is a triple² $(Rel, TupleID, Column)$ denoting the *Column*-field of tuple $TupleID$ in database relation Rel . Instead of having a variable number of component relations, possibly with different arities, we store all values in a single big relation C that has a fixed schema. In this representation we need a restricted flavor of world-ids called *local world-ids* (LWIDs). The local world-ids refer only to the possible worlds within one component and do not bring the aforementioned drawbacks of “global” world IDs.

Given a WSD (C_1, \dots, C_m) with schemas $\hat{C}_i[U_i]$ (we now distinguish between the relation and its name), we populate the corresponding UWSD as follows.

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

Remark 4.3.1 In general, the value column in the component relation C must store values for fields of different type. However this can be easily dealt with. One possibility is to store all values as strings and explicitly use casts when performing an operation on the values. Alternatively, one could have one component relation for each data type. In both cases the schema remains fixed.

Example 4.3.2 Let us consider again the world-set decomposition from Figure 1.3. We assume that the local world IDs \hat{s} are the indexes of tuples in component relations in the order given by Figure 1.3. Using our alternative representation with fixed relational schemata for F , W and C , the decomposition is as shown in Figure 4.2. \square

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

²That is, FID really takes three columns, but for readability we keep them together under a common name in this Chapter.

C	FID	LWID	VAL	F	FID	CID
	(R, t_1, S)	1	185		(R, t_1, S)	C_1
	(R, t_1, S)	2	785		(R, t_1, N)	C_2
	(R, t_1, S)	3	785		(R, t_1, M)	C_3
	(R, t_2, S)	1	186		(R, t_2, S)	C_1
	(R, t_2, S)	2	185		(R, t_2, N)	C_4
	(R, t_2, S)	3	186		(R, t_2, M)	C_5
	(R, t_1, N)	1	Smith	W	C_1	1
	(R, t_1, M)	1	1		C_1	2
	(R, t_1, M)	2	2		C_1	3
	(R, t_1, M)	2	2		C_2	1
	(R, t_2, N)	1	Brown		C_3	1
	(R, t_2, M)	1	1		C_3	2
	(R, t_2, M)	2	2		C_4	1
	(R, t_2, M)	3	3		C_5	1
	(R, t_2, M)	4	4		C_5	2
					C_5	3
					C_5	4

Figure 4.2: A uniform WSD for our running example.

R^0	S	N	M	F	FID	CID
t_1	?	Smith	?			(R, t_1, S)
t_2	?	Brown	3		(R, t_1, M)	C_2
					(R, t_2, S)	C_1
C	FID	LWID	VAL	W	CID	LWID
	(R, t_1, S)	1	185			C_1
	(R, t_2, S)	1	186		C_1	2
	(R, t_1, S)	2	785		C_1	3
	(R, t_2, S)	2	185		C_2	1
	(R, t_1, S)	3	785		C_2	2
	(R, t_2, S)	3	186			
	(R, t_1, M)	1	1			
	(R, t_1, M)	2	2			

Figure 4.3: A UWSDT corresponding to the WSDT of Figure 4.1.

Example 4.3.3 Consider the example of Figure 4.3, which is the uniform version of the WSDT of Figure 4.1. 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 '?'. Just as before, the possible values for the placeholders are defined in the component table C . In practice, for incomplete-information databases, 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. \square

It is again easy to verify that

Proposition 4.3.4 *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*.

Chapter 5

Queries on Decompositions

In this chapter 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 in both representation systems. For that reason we will first concentrate on query evaluation in the WSD framework. Query evaluation for UWSDTs is then discussed in Chapter 6.

For each relational algebra query Q , we provide 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.

Example 5.0.5 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 $\sigma_{A=1}(R) \cup \sigma_{B=2}(R)$. \square

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 relation 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 column $R.t.A_i$:

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

Then $\text{copy}(R, P)$ executes

$$C := \text{ext}(C, R.t_i.A, P.t_i.A)$$

for each component C of our WSD and each $R.t_i.A \in \mathcal{S}(C)$.

For each relational algebra operation, the input WSD is *extended* by the result of the operation. Note that in our examples, we will consistently just show the decomposition of the result relation. The reason for this is simply our attempt to keep the WSDs readable.

We next present implementation of the relational algebra operations on WSDs.

5.1 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 of component relations may neither contain any world-tuple $t \in R^A$ in its entirety nor will each component tuple in general represent (parts of) only one world-tuple.

Example 5.1.1 We use the set of eight worlds over the relation R of Figure 5.1 and its maximal 7-WSD of Figure 5.1 (b) as a running example for most of this Chapter. The second component (from the left) of the WSD spans over several tuples and attributes. It contains the values for $R.t_1.B$, $R.t_1.C$, and $R.t_2.B$, i.e. some but not all of the attributes of the first and second tuple of R^A . Each of the remaining six components refers to one tuple and one attribute. \square

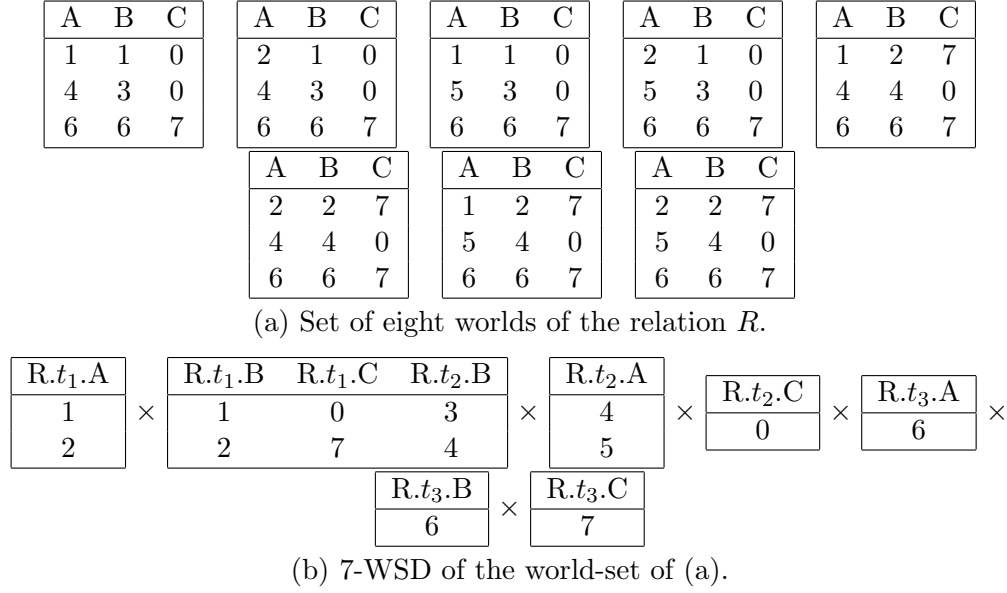


Figure 5.1: World-set and its decomposition

Algorithm 1 Selection with constant

```

1: procedure SELECT[ $A\theta c$ ] //compute  $P := \sigma_{A\theta c}R$ 
2:   COPY(R,P);
3:   for all  $1 \leq i \leq |P|_{max}$  do
4:     Let  $C$  be the component of  $P.t_i.A$ ;
5:     for all  $t_C \in C$  do
6:       if not  $(t_C.(P.t_i.A) \theta c)$  then
7:          $t_C.(P.t_i.A) := \perp$ ;
8:       end if
9:     end for
10:    PROPAGATE( $P.t_i.A$ );
11:  end for
12: end procedure
    
```

Thus a selection in general 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 1 checks for each tuple t_i of P and for each tuple t_C in the component C that has attribute $P.t_i.A$ whether $t_C.(P.t_i.A)$ satisfies the selection condition¹, i.e., $t_C.(P.t_i.A)\theta c$. If

¹Of course the tuples over P can be different – even their number may vary – in each of the possible worlds over P . However, since world-set relations, and therefore their

Algorithm 2 Propagation of \perp -values

```

1: procedure PROPAGATE( $R.t_i.A$ ) //Propagate  $\perp$ -values
2:   Let  $C$  be the component of  $R.t_i.A$ ;
3:   for all  $t_C \in C$  do
4:     if  $t_C.(R.t_i.A) = \perp$  then
5:       for all  $A'$  such that  $P.t_i.A' \in \mathcal{S}(C)$  do
6:          $t_C.(P.t_i.A') := \perp$ ;
7:       end for
8:     end if
9:   end for
10: end procedure
    
```

it does not, $t_C.(P.t_i.A)$ is assigned value \perp . This marks the tuple $P.t_i$ in all worlds that take values from t_C as deleted. Moreover, for all other attributes A' such that $P.t_i.A'$ is an attribute of the same component C , we set $t_C.(P.t_i.A') := \perp$ (see Algorithm 2). This assures that if we later project away $P.A$, we do not erroneously “reintroduce” tuple $P.t_i$ into worlds that take values from t_C . We will revisit this issue again later.

$$\begin{array}{|c|} \hline P.t_1.A \\ \hline 1 \\ \hline 2 \\ \hline \end{array} \times \begin{array}{|c|c|c|c|} \hline P.t_1.B & P.t_1.C & P.t_2.B & \\ \hline \perp & \perp & 3 & \\ \hline 2 & 7 & 4 & \\ \hline \end{array} \times \begin{array}{|c|} \hline P.t_2.A \\ \hline 4 \\ \hline 5 \\ \hline \end{array} \times \begin{array}{|c|} \hline P.t_2.C \\ \hline \perp \\ \hline \end{array} \times \begin{array}{|c|} \hline P.t_3.A \\ \hline 6 \\ \hline \end{array} \times \\
 \begin{array}{|c|} \hline P.t_3.B \\ \hline 6 \\ \hline \end{array} \times \begin{array}{|c|} \hline P.t_3.C \\ \hline 7 \\ \hline \end{array}$$

 (a) $P := \sigma_{C=7}(R)$ applied to the WSD of Figure 5.1 (b).

$$\begin{array}{|c|} \hline P.t_1.A \\ \hline 1 \\ \hline 2 \\ \hline \end{array} \times \begin{array}{|c|c|c|c|} \hline P.t_1.B & P.t_1.C & P.t_2.B & \\ \hline 1 & 0 & \perp & \\ \hline \perp & \perp & \perp & \\ \hline \end{array} \times \begin{array}{|c|} \hline P.t_2.A \\ \hline 4 \\ \hline 5 \\ \hline \end{array} \times \begin{array}{|c|} \hline P.t_2.C \\ \hline 0 \\ \hline \end{array} \times \begin{array}{|c|} \hline P.t_3.A \\ \hline 6 \\ \hline \end{array} \times \\
 \begin{array}{|c|} \hline P.t_3.B \\ \hline \perp \\ \hline \end{array} \times \begin{array}{|c|} \hline P.t_3.C \\ \hline 7 \\ \hline \end{array}$$

 (b) $P := \sigma_{B=1}(R)$ applied to the WSD of Figure 5.1 (b).

 Figure 5.2: Selections $P := \sigma_{C=7}(R)$ and $P := \sigma_{B=1}(R)$ with R from Figure 5.1 (b).

decompositions, reserve a slot for the same $|P|_{max}$ tuples in each world, and just some worlds may have some of these tuples marked as invalid, we may just as well refer to a tuple of P as an object that has a different *value*, and in some cases \perp , in each possible world.

Example 5.1.2 The answer P to $\sigma_{C=7}(R)$ is represented by the WSD of Figure 5.2 (a). Figure 5.2 (b) shows the result of query $\sigma_{B=1}(R)$. Note that the resulting WSDs should contain both the query answer P and the original relation R , but for clarity we only show the representation of P . One can observe that for both results in Figure 5.2 we may obtain worlds of different sizes. For example the worlds that take values from the first tuple of the second component relation 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

5.2 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 two attributes of a tuple are accessed, which do not necessarily reside in the same component.

The 7-WSD of Figure 5.1 (b) has no component containing values for both the attributes A and B of any tuple of the decomposed worlds. Therefore, the test of the join condition for each of these tuples has to span over several components containing values for A and B . Additionally, the join condition may exclude some possible combinations of the values from different components. Therefore, the current decomposition may not capture exactly the combinations of values satisfying the join condition. It may then be necessary to compose into one component the components that have values for A and B within a same tuple of the decomposed worlds. After the composition phase, the selection algorithm follows the pattern of the selection with constant.

Example 5.2.1 Consider the query $\sigma_{A=B}(R)$, where R is represented by the 7-WSD of Figure 5.1 (b). Figure 5.3 (a) shows the 6-WSD obtained from the one of Figure 5.1 (b) after tuple t_1 has been processed (requiring the composition of the first and second component of the WSD of Figure 5.1 (b)). Further applying the selection to tuple t_2 yields the 4-WSD of Figure 5.3 (b). Finally, processing tuple t_3 leads to the composition of the second and the third components, as shown in Figure 5.3 (c). This 4-WSD represents five worlds, where one world has three tuples, three worlds have two tuples each, and one world has one tuple. \square

Algorithm 5.2 implements the selection with a join condition.

P.t ₁ .A	P.t ₁ .B	P.t ₁ .C	P.t ₂ .B
1	1	0	3
⊥	⊥	⊥	3
⊥	⊥	⊥	4
2	2	7	4

P.t ₂ .A
4
5

P.t ₂ .C
0

P.t ₃ .A
6

P.t ₃ .B
6

P.t ₃ .C
7

(a) 6-WSD after filtering tuple t_1 using condition $A = B$.

P.t ₁ .A	P.t ₁ .B	P.t ₁ .C	P.t ₂ .A	P.t ₂ .B
1	1	0	⊥	⊥
⊥	⊥	⊥	⊥	⊥
⊥	⊥	⊥	4	4
2	2	7	4	4
2	2	7	⊥	⊥

P.t ₂ .C
0

P.t ₃ .A
6

P.t ₃ .B
6

P.t ₃ .C
7

(b) 5-WSD after filtering tuples t_1, t_2 using condition $A = B$.

P.t ₁ .A	P.t ₁ .B	P.t ₁ .C	P.t ₂ .A	P.t ₂ .B
1	1	0	⊥	⊥
⊥	⊥	⊥	⊥	⊥
⊥	⊥	⊥	4	4
2	2	7	4	4
2	2	7	⊥	⊥

P.t ₂ .C
0

P.t ₃ .A	P.t ₃ .B
6	6

P.t ₃ .C
7

(c) 5-WSD after filtering all tuples using condition $A = B$

Figure 5.3: $P = \sigma_{A=B}(R)$ with R from Figure 5.1 (b).

Algorithm 3 Selection with a join condition

```

1: procedure SELECT[ $A\theta B$ ] //compute  $P := \sigma_{A\theta B}R$ 
2:   COPY(R,P);
3:   for all  $1 \leq i \leq |P|_{max}$  do
4:     Let  $C$  be the component of  $P.t_i.A$ ;
5:     Let  $C'$  be the component of  $P.t_i.B$ ;
6:     if  $C \neq C'$  then
7:       replace components  $C, C'$  by  $C := C \times C'$ ;
8:     end if
9:     for all  $t_C \in C$  do
10:      if not( $t_C.(P.t_i.A) \theta t_C.(P.t_i.B)$ ) then
11:         $t_C.(P.t_i.A) := \perp$ ;
12:         $t_C.(P.t_i.B) := \perp$ ;
13:      end if
14:    end for
15:    PROPAGATE( $P.t_i.A$ );
16:    PROPAGATE( $P.t_i.B$ );
17:  end for
18: end procedure
    
```

5.3 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.

This works well for the simple case where no \perp -values are involved.

$$\begin{array}{|c|c|} \hline P.t_1.B & P.t_2.B \\ \hline 1 & 3 \\ 2 & 4 \\ \hline \end{array} \times \begin{array}{|c|} \hline P.t_3.B \\ \hline 6 \\ \hline \end{array}$$

Figure 5.4: $P := \pi_B(R)$ with R from Figure 5.1 (b).

Example 5.3.1 The WSD of Figure 5.4 shows the relation $P := \pi_B(R)$, where R is represented by the WSD of Figure 5.1 (b). \square

However, these operations prove to be insufficient for the correctness of the algorithm when the WSD contains \perp -values. Consider for example the world-set decomposition in Figure 5.5 (a). We may assume that this is the

result of a selection that introduced \perp -values in some fields of the first component. The WSD represents four worlds, where no world contains both t_1 and t_2 . Projecting away the A column would remove the first component and the information that t_1 and t_2 do not appear together in the same world will be lost. One solution is to merge all components involving tuples t_1 and t_2 and use the already described propagation of \perp values. However, we only need to find one remaining component containing information about the two tuples. For example in Figure 5.5 (b) we only merge the components for $t_1.B$ and $t_2.B$ to produce a correct result.

Algorithm 4 implements the projection operation on WSDs. After creating a copy of the initial relation R , it propagates the \perp -values (lines 4-18) by finding an appropriate component to carry on the information, as we already discussed. Note that we do not specify how to find such a component. Ideally, it will contain only a single column and have minimal size, so that the result of the composition remains small. At the end all projected out columns are dropped from the respective components.

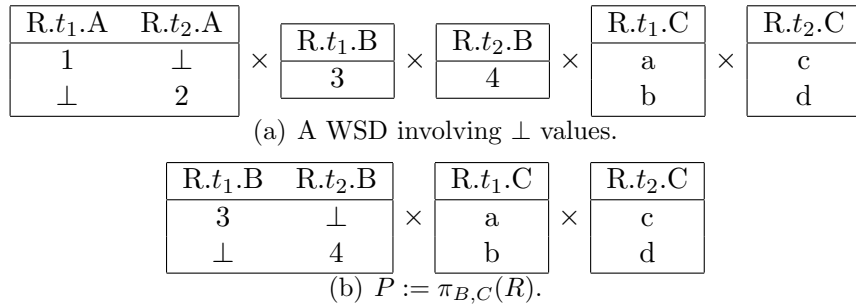


Figure 5.5: Evaluating projection when \perp -values are involved.

Remark 5.3.2 Note that in the implementation of the selection operators we already cover the simpler case where for a projected away attribute A and a tuple t , there is at least one remaining attribute B such that $t.A$ and $t.B$ are defined in the same component. We do this with Algorithm 2; for a component C and a field $R.t.A$ in C the algorithm propagates \perp -values to all $R.t$ -fields defined in C. Therefore in line 7 of Algorithm 4 we check whether the component contains a field from the same tuple which is in the projection list. Only in the negative case we compose the component with another one to preserve the needed information. \square

Algorithm 4 Projection

```

1: procedure PROJECT[ $U$ ] //compute  $P := \pi_U R$ 
2:   COPY( $R, P$ );
3:   //Propagate  $\perp$ -s
4:   repeat
5:     for all  $1 \leq i \leq |P|_{max}$  and  $A \notin U$  do
6:       Let  $C$  be the component of  $P.t_i.A$ ;
7:       if  $\neg \exists B$  such that  $B \in U, P.t_i.B \in S(C)$  then
8:         for all  $B$  such that  $P.t_j.B \in S(C)$  do
9:           if  $\forall t_c \in C: t_c.(P.t_i.A) = \perp$  or  $t_c.(P.t_j.B) = \perp$  then
10:            Let  $C'$  be the component of  $P.t_i.A'$  for some  $A' \in U$ ;
11:            Replace  $C, C'$  by  $C := C \times C'$ ;
12:            PROPAGATE( $P.t_i.A$ );
13:            Project away  $P.t_i.A$  from  $C$ ;
14:          end if
15:        end for
16:      end if
17:    end for
18:  until a fixpoint is reached

19:  //Drop projected out columns
20:  for all  $1 \leq i \leq |P|_{max}$  and  $A \notin U$  do
21:    let  $C$  be the component of  $P.t_i.A$ ;
22:    Project away  $P.t_i.A$  from  $C$ ;
23:  end for
24: end procedure

```

Algorithm 5 Product

```

1: procedure PRODUCT //compute  $T := R \times S$ 
2:   for all  $1 \leq j \leq |S|_{max}$  and  $R.t_i.A \in \mathcal{S}(R)$  do
3:     let  $C$  be the component of  $R.t_i.A$ ;
4:      $C := \text{EXT}(C, R.t_i.A, T.t_{ij}.A)$ ;
5:   end for
6:   for all  $1 \leq i \leq |R|_{max}$  and  $S.t_j.A \in \mathcal{S}(S)$  do
7:     let  $C'$  be the component of  $S.t_j.A$ ;
8:      $C' := \text{EXT}(C', S.t_j.A, T.t_{ij}.A)$ ;
9:   end for
10: end procedure
    
```

5.4 Product

The product $T := R \times S$ of two relations R and S , which have disjoint attribute sets and are represented by a WSD \mathcal{C} 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 i th (j th) copy is named $T.t_{ij}.A$ if the original has name $R.t_i.A$ or $S.t_j.A$.

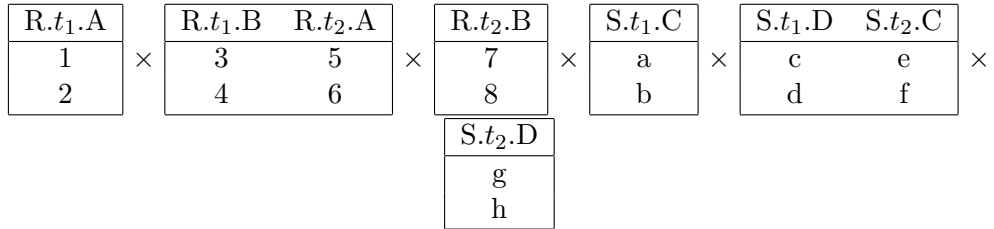
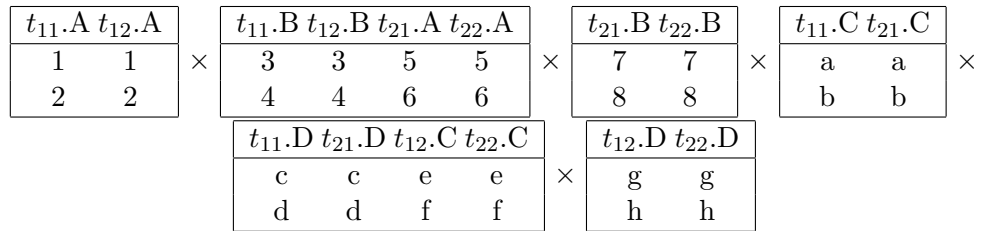

 (a) WSD of two relations R and S .

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

 Figure 5.6: The product operation $R \times S$.

Example 5.4.1 Figure 5.6 (b) shows the WSD for the product of relations R and S represented by the WSD of Figure 5.6 (a). \square

5.5 Union

Algorithm 6 for computing the union $T := R \cup S$ of two relations R and S works similarly to those for projection and product. Each component C containing values of R or S is extended such that all values of R and S become also values of T in the respective world. Figure 5.7 shows the resulting WSD for the union of the relations R and S in Figure 5.6 (a).

Algorithm 6 Union

```

1: procedure UNION //compute  $T := R \cup S$ 
2:   for all  $1 \leq i \leq |R|_{max}$  and  $A \in \mathcal{S}(R)$  do
3:     let  $C$  be the component of  $R.t_i.A$ ;
4:      $C := \text{EXT}(C, R.t_i.A, T.(R.t_i).A)$ ;
5:   end for
6:   for all  $1 \leq j \leq |S|_{max}$  and  $A \in \mathcal{S}(S)$  do
7:     let  $C'$  be the component of  $S.t_j.A$ ;
8:      $C' := \text{EXT}(C', S.t_j.A, T.(S.t_j).A)$ ;
9:   end for
10: end procedure
    
```

R.t ₁ .A	T.(R.t ₁).A	×	R.t ₁ .B	R.t ₂ .A	T.(R.t ₁).B	T.(R.t ₂).A	×
1	1		3	5	3	5	
2	2	4	6	4	6		
		×	R.t ₂ .B	T.(R.t ₂).B			×
			7	7	S.t ₁ .C	T.(S.t ₁).C	
			8	8	a	a	
					b	b	
S.t ₁ .D	S.t ₂ .C	T.(S.t ₁).D	T.(S.t ₂).C	×	S.t ₂ .D	T.(S.t ₂).D	
c	e	c	e		g	g	
d	f	d	f	h	h		

Figure 5.7: The union $T = R \cup S$ for the relations in Figure 5.6 (a).

5.6 Renaming

The operation $\delta_{A \rightarrow A'}(R)$, which renames attribute A or relation R to A' , is very easy to implement. For each tuple id t , let C be the component that has the attribute $R.t.A$. Then we rename this attribute to $R.t.A'$ as $C := \delta_{R.t.A \rightarrow R.t.A'}(C)$. Algorithm 7 implements renaming on WSDs.

Algorithm 7 Rename

```

1: procedure RENAME //compute  $\delta_{A \rightarrow A'}(R)$ 
2:   for all  $1 \leq i \leq |R|_{max}$  do
3:     let  $C$  be the component of  $R.t_i.A$ ;
4:      $C := \delta_{R.t_i.A \rightarrow R.t_i.A'}(C)$ ;
5:   end for
6: end procedure

```

5.7 Difference

To demonstrate the challenges for implementing the difference operation, let us consider the following example. Figure 5.8 shows the world-set decomposition of two single-attribute relations R and S . Let us compute the query $R - S$. The first tuple of R is in the result only if $R.t_1.A \neq S.t_1.A$. This requires us to compose the components for $R.t_1.A$ and $S.t_1.A$ and is illustrated in Figure 5.9 (a). Ensuring that $R.t_2.A \neq S.t_2.A$ leads to further component composition. Figure 5.9 (a) shows the final result for the difference operation.

R.t ₁ .A	×	R.t ₂ .A	×	S.t ₁ .A
1		2		1
2		3		3

Figure 5.8: WSD of two relations R and S .

As seen in the example, the difference operation $R - S$ may require the merging of all the components that handle fields of R or S and is thus by far the least efficient to implement. However, this is a known problem of strong representation systems for relational algebra. For example, it is also known that executing the difference operation on c-tables takes exponential time in general [14].

5.8 Discussion

The algorithms presented in this chapter provide us with a machinery to execute any relational algebra query using the operations σ , π , \times , and \cup . Most operations can be expressed as relational algebra queries themselves. However, the encoding in relational algebra is nonuniform in that each WSD may have a different schema. At the same time, one can easily derive algorithms which generate encodings of the operations in relational algebra on WSDs.

R.t ₁ .A	S.t ₁ .A	T.t ₁ .A		R.t ₂ .A
1	1	⊥	×	2
1	3	1		3
2	1	2		
2	3	2		

(a) Partial result for $T = R - S$ after processing $R.t_1$.

R.t ₁ .A	S.t ₁ .A	T.t ₁ .A	R.t ₂ .A	T.t ₂ .A
1	1	⊥	2	2
1	1	⊥	3	3
1	3	1	2	2
1	3	1	3	⊥
2	1	2	1	⊥
2	1	2	3	3
2	3	2	2	2
2	3	2	3	⊥

(b) Final result for $T = R - S$ after processing $R.t_2$.

 Figure 5.9: The difference operation $R - S$.

The only exception here are the selection with “join conditions” $A\theta B$ and the difference operation $R - S$. A *fixed* such query can require the merging of an arbitrary number of component relations into a single component, causing an exponential blowup in the size of the WSD. Since relational algebra has polynomial-time data complexity [1], i.e., fixed queries can only produce results of size polynomial in the input data, selection with conditions $A\theta B$ *cannot* be expressible in relational algebra.²

Given a relational algebra query Q , let \hat{Q} denote the query processor on WSDs we obtain by replacing each operation of Q by its corresponding operation on WSDs.

Theorem 5.8.1 (Correctness) *Let \mathcal{W} be a WSD and let \mathcal{W}' be the WSD obtained from $\hat{Q}(\mathcal{W})$ by dropping all relations but the result relation of \hat{Q} . Then,*

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

²For that matter, selections $\sigma_{A\theta B}$ are also not expressible in more expressive query languages with polynomial-time data complexity, such as datalog with stratified negation. As it turns out, the ability to create new identifiers is necessary to form exponential sets of tuples.

5.9 Simplifying WSDs

Sometimes, when our input is an optimally decomposed WSD, the WSDs obtained by the algorithms in this chapter are not maximal anymore. Figure 5.10 gives three condition-action rules for simplifying WSDs. A condition-action rule has the form $\frac{\text{condition}}{\text{action}}$ and expresses that if the *condition* holds, then the *action* is performed.

$$\frac{C \in \mathcal{C} \wedge C = C_1 \times C_2}{\mathcal{C} := \mathcal{C} - \{C\} \cup \{C_1, C_2\}} \quad (5.1)$$

$$\frac{C_1, C_2 \in \mathcal{C} \wedge X.A \in \mathcal{S}(C_1) \wedge X.B \in \mathcal{S}(C_2) \wedge \pi_{X.A}(C_1) = \{\perp\}}{C_2 := \pi_{\mathcal{S}(C_2) - \{X.B\}}(C_2)} \quad (5.2)$$

$$\frac{C \in \mathcal{C} \wedge X.A \in \mathcal{S}(C) \wedge \pi_{X.A}(C) = \{\perp\} \wedge \forall C' \forall B C' \in \mathcal{C} \wedge A \neq B \Rightarrow X.B \notin \mathcal{S}(C')}{C := \pi_{\mathcal{S}(C) - \{X.A\}}(C)} \quad (5.3)$$

Figure 5.10: WSD Simplification Rules.

The first rule allows to decompose a component that consists of two independent parts. The second and third rule together allow to remove tuples that are invalid in all worlds (X stands for $P.t$, where P is a relation name and t is a tuple identifier).

Example 5.9.1 The WSD of Figure 5.2 (a) exhibits an interesting case of redundancy, given by the presence of only \perp values for $P.t_2.C$. This can be interpreted as the absence of tuple t_2 of P from all worlds, and allows us to remove the entries of that tuple from all components. Rules (5.2) and (5.3) transform this WSD into the equivalent WSD of Figure 5.11. \square

Example 5.9.2 The 4-WSD of Figure 5.3 (c) admits the equivalent 5-WSD of Figure 5.3 (b), where the second component is decomposed into two components. Such a case of non-maximality can be corrected by Rule (5.1). \square

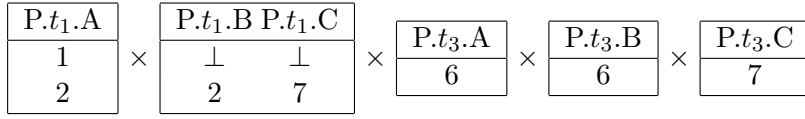


Figure 5.11: Simplification of WSD of Figure 5.2 (a).

Remark 5.9.3 Note that the non-maximality case of the world-set decomposition from Figure 5.3 (c) cannot appear for UWSDTs, because all but the first component of Figure 5.3 (c) contain only one tuple and are stored in the template relation, where no component merging occur. \square

Chapter 6

Efficient Query Evaluation on UWSDTs

The algorithms for computing the relational operations on WSDs presented in Chapter 5 can be easily adapted to UWSDTs. To do this, we follow closely the mapping of WSDs, represented as sets of components \mathcal{C} , to equivalent UWSDTs, represented by a triple (F, C, W) and at least one template relation R^0 :

- Consider a component K of WSD \mathcal{C} having an attribute $R.t.A$ with a value v . In the equivalent UWSDT, this value can be stored in the template relation R^0 if v is the only value of $R.t.A$, or in the component C otherwise. In the latter case, the template R^0 contains the placeholder $R.t.A$ in the tuple t . In addition, in the mapping relation F there is an entry with the placeholder $R.t.A$ and a component identifier c , and C contains a tuple formed by $R.t.A$, the value v and a world identifier w .
- Worlds of different sizes are represented in WSDs by allowing \perp values in components, and in UWSDTs by allowing for a same placeholder different amount of values in different worlds.

Next we demonstrate how queries can be evaluated in the context of UWSDTs.

An important operation is the composition of components. In WSDs this is accomplished via the normal relational product operation on the two operands. Now, as the values for all components are defined in the same relation, the product must be simulated through insertion of tuples. To illustrate this let us consider the following example. Figure 6.1 shows two components from a WSD that have 2 and 3 worlds, respectively, and the result of their composition. One can observe that the resulting component

C_1	R.t ₁ .A	R.t ₂ .A
	a	b
	c	d

×

C_2	R.t ₂ .B
	1
	2
	3

(a) Two components

$C_1 \times C_2$	R.t ₁ .A	R.t ₂ .A	R.t ₂ .B
	a	b	1
	a	b	2
	a	b	3
	c	d	1
	c	d	2
	c	d	3

(b) Product of the components in (a)

Figure 6.1: Composing components in WSDs.

contains three copies of each tuple from the C_1 , which is exactly the number of tuples in C_2 . Accordingly, each record from C_2 has been copied twice in the result. Figure 6.2 shows the same computation in the context of UWSDTs.

Algorithm 6 gives our efficient implementation of the selection with constant.

Algorithm 8 Selection with constant on UWSDTs.

```

1: procedure SELECT[ $A\theta c$ ] //compute  $P := \sigma_{A\theta c}R$ 
2:    $P^0 := \sigma_{A\theta c \vee A=?}(R^0)$ ;
3:    $F := F \cup \{(P.t.B, k) \mid (R.t.B, k) \in F, t \in P^0\}$ ;
4:    $C := C \cup \{(P.t.B, w, v) \mid (R.t.B, w, v) \in C, t \in P^0, (B = A \Rightarrow v\theta c)\}$ ;
5:    $C := C - \{(P.t.X, w, v) \in C \mid (P.t.X, k), (P.t.Y, k) \in F, t \in P^0,$ 
6:      $X \neq Y, \nexists v' : (P.t.Y, w, v') \in C\}$ ;
7:    $F := F - \{(P.t.B, k) \mid (P.t.B, k) \in F, \nexists w, v : (P.t.B, w, v) \in C\}$ ;
8:    $P^0 := P^0 - \{t \mid t \in P^0, \nexists B, a : (P.t.B, a) \in F\}$ ;
9:    $W := \pi_{cid, lwid}(F \bowtie C)$ ;
10: end procedure
    
```

In contrast to some algorithms on WSDs, in UWSDTs we do not create a copy P of R at the beginning, but rather compute directly P from R using standard relational algebra operators. The template P^0 is initially the set of tuples of R^0 that satisfy the selection condition, or that have a placeholder '?' for the attribute A (line 1). We extend the mapping relation F with placeholders of P^0 (line 2), and the component relation C with the values

			C	FID	LWID	VAL
				(R,t ₁ ,A)	1	a
F	FID	CID		(R,t ₁ ,A)	2	c
	(R,t ₁ ,A)	C ₁		(R,t ₂ ,A)	1	b
	(R,t ₂ ,A)	C ₁		(R,t ₂ ,A)	2	d
	(R,t ₂ ,B)	C ₂		(R,t ₂ ,B)	1	1
				(R,t ₂ ,B)	2	2
				(R,t ₂ ,B)	3	3

(a) Mapping and component relation for the components in Figure 6.1 (a)

			C	FID	LWID	VAL
				(R,t ₁ ,A)	1	a
				(R,t ₁ ,A)	2	a
				(R,t ₁ ,A)	3	a
				(R,t ₁ ,A)	4	c
				(R,t ₁ ,A)	5	c
				(R,t ₁ ,A)	6	c
F	FID	CID		(R,t ₂ ,A)	1	b
	(R,t ₁ ,A)	C ₁ × C ₂		(R,t ₂ ,A)	2	b
	(R,t ₂ ,A)	C ₁ × C ₂		(R,t ₂ ,A)	3	b
	(R,t ₂ ,B)	C ₁ × C ₂		(R,t ₂ ,A)	4	d
				(R,t ₂ ,A)	5	d
				(R,t ₂ ,A)	6	d
				(R,t ₂ ,B)	1	1
				(R,t ₂ ,B)	4	1
				(R,t ₂ ,B)	2	2
				(R,t ₂ ,B)	5	2
				(R,t ₂ ,B)	3	3
				(R,t ₂ ,B)	6	3

(b) Product of the components in (a)

Figure 6.2: Composing components in UWSDTs.

of P^0 placeholders, where the values of placeholders $P.t.A$ for the attribute A must satisfy the selection condition (line 3). If a placeholder $P.t.A$ has no value satisfying the selection condition, then t is removed from P^0 (line 6) and all placeholders of t are removed from F (line 5) together with their values from C (line 4).

Many of the standard query optimization techniques are also applicable in our context. For our experiments reported in Chapter 8, we performed the following optimizations. For the evaluation of a query involving join,

we interleave the product and the selections with join conditions. When evaluating a query involving several selections and projections on the same relation, we again merge these operators and perform the steps of Algorithm 6 only once.

Example 6.0.4 To evaluate $P := \pi_{A,C,D}(\sigma_{A>0 \vee D=0}R)$, we need the following minor extensions to Algorithm 6. First, the selection condition becomes our condition disjunct $A > 0 \vee D = 0$ augmented with checks on whether A or D contain placeholders. Second, we incorporate the projection in the computation of P^0 (line 1), add to F only the placeholders of P^0 for the attributes in the projection list (line 2), and add to C only the values of these placeholders (line 3). \square

Chapter 7

Chasing Dependencies

In this chapter we address the problem of removing inconsistent worlds in an incompletely specified database. We present a method called *Chase* [4, 22, 1] in the spirit of the work of [15] for data cleaning on a world-set decomposition of a relation R , given a set of dependencies Φ .

We consider the following types of dependencies over a relation R : *functional dependencies* denoted by

$$A_1, \dots, A_m \rightarrow A_0$$

where $A_i \in S(R)$, $0 \leq i \leq m$, and *equality-generating dependencies* of the form

$$\phi_1 \wedge \phi_2 \wedge \dots \wedge \phi_m \Rightarrow \phi_0$$

where each $\phi_i(A_i) = A_i \theta_i c_i$, $0 \leq i \leq m$ is a binary operator comparing the value of an attribute $A_i \in S(R)$ with a constant c_i . Relation R satisfies an equality-generating dependency *egd* (denoted by $R \models \text{egd}$) if for each tuple $t \in R$

$$t.A_1 \theta_1 c_1 \wedge \dots \wedge t.A_m \theta_m c_m \Rightarrow t.A_0 \theta_0 c_0$$

Recall Example 1.0.2 from the introduction. The uniqueness constraint for the social security number is a functional dependency $S \rightarrow N, M$, which is of course equivalent to the two functional dependencies $S \rightarrow N$ and $S \rightarrow M$. To enforce this constraint we combined the two S fields ($t_1.S$ and $t_2.S$) in the same component and removed the worlds in which both have the same value (see Figure 1.3).

Assume now that from a reliable source we have the information that the person with social security number 785 is married. The current decomposition allows invalid combinations of values: those worlds in which $t_1.S = 785$ and $t_1.M \neq 1$ (1 is the code for married). To remove inconsistencies, we must compose the first and the third components and remove from the new

component all tuples that do not satisfy the given dependency. As a result of this data-cleaning step we obtain the 4-WSD in Figure 7.1.

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

Figure 7.1: Result of chasing $S = 785 \Rightarrow M = 1$ on the WSD in Figure 1.3.

One would probably observe that enforcing a dependency on a WSD resembles the selection operation with condition $A\theta B$ presented in Chapter 5. In both cases we identify dependencies across components and compose dependent components. Nevertheless there is an important difference between the two operations. In the selection operation we are interested in finding *a subset of the tuples* valid in *some world*. If a tuple fails to satisfy the selection condition, it is simply dropped from the result. On the other hand, when enforcing dependencies on a WSD, we want to get *a subset of the possible worlds* such that the dependencies hold for *all tuples*. If a tuple has no valid values in any of the worlds, this automatically means that the database is inconsistent with respect to the given set of dependencies.

As seen in the previous examples, cleaning inconsistent worlds involves two basic steps: (1) composing dependent components into one and (2) removing inconsistent tuples from the resulting component. Repeating these two steps iteratively for each dependency and each tuple in the given WSD \mathcal{W} would result in a WSD \mathcal{W}' satisfying all constraints.

Before proceeding to the formal algorithm for chasing dependencies, we introduce the following notations.

If

$$fd = A_1, \dots, A_m \rightarrow A_0$$

is a functional dependency, $s, t \in \mathcal{W}$ and all attributes $s.A_i, t.A_i, 0 \leq i \leq m$ are defined in a component C , the operation that retrieves all worlds in C satisfying fd can be expressed by

$$\sigma_{C.(s.A_1) \neq C.(t.A_1) \vee \dots \vee C.(s.A_m) \neq C.(t.A_m) \vee C.(s.A_0) = C.(t.A_0)}(C)$$

where σ is the relational selection. For the sake of brevity we write $\sigma_{fd}(C)$. Similarly, if t is a tuple from \mathcal{W} ,

$$egd = A_1\theta_1c_1 \wedge \dots \wedge A_m\theta_m c_m \Rightarrow A_0\theta_0c_0$$

is an equality-generating dependency and all attributes $t.A_i, 0 \leq i \leq m$ are defined in a component C , the operation

$$\sigma_{\neg(C.(t.A_1)\theta_1c_1)\vee\dots\vee\neg(C.(t.A_m)\theta_m c_m)\vee(C.(t.A_0)\theta_0c_0)}(C)$$

retrieves all valid worlds (with respect to egd) from C . We write shortly $\sigma_{egd}(C)$.

The algorithm of Figure 9 implements the data cleaning for a given world-set decomposition and a set of dependencies Φ . Note that as opposed to the traditional chase on tableaux ([22]), here we do not need a fixpoint computation but a single scan over all dependencies and tuples in the WSD. The reason for this is that enforcing a functional or equality-generating dependency on a WSD cannot induce further inconsistencies in the data.

We can further refine the data cleaning rules and avoid redundant operations if we make the following observations. For a functional dependency

$$fd = A_1, \dots, A_m \rightarrow A_0$$

and tuples s and t , if for an attribute $A_i, 1 \leq i \leq m$ it holds that $s.A_i = t.A_i$ in all worlds, we do not need to join the components defining $s.A_i$ and $t.A_i$. Alternatively, if in all worlds $s.A_0 \neq t.A_0$, we can leave the components for $s.A_0$ and $t.A_0$ unmerged. The same idea can be applied for an equality-generating dependency

$$egd = A_1\theta_1c_1 \wedge \dots \wedge A_m\theta_m c_m \Rightarrow A_0\theta_0c_0$$

tuple s and an attribute $A_i, 1 \leq i \leq m$, such that $\phi_i(t.A_i) = true$ in all worlds, or $\phi_0(t.A_0)$ is always *false*, we do not need to compose the corresponding component.

In general the order in which dependencies are chased has an impact on the size of the resulting decomposition. This means that the world-set decomposition produced by the Chase algorithm may be non-maximal, which was also the case with querying. Consider for example the WSD in Figure 7.2 (a) and the set of two dependencies

$$D = \{d_1 = B \rightarrow C, d_2 = A = 1 \Rightarrow B \neq 2\}$$

Chasing $d_1 = B \rightarrow C$ requires the compositions of the components for $t_1.B, t_2.B, t_1.C$ and $t_2.C$, and enforcing d_2 deletes tuples from the resulting component (see Figure 7.2 (a)). However, if we start with d_2 , we will benefit from the side effect of cleaning inconsistent worlds with respect to d_1 and no merging of component will be necessary (Figure 7.2 (b)). Note that although the two world-set decompositions are different, they are equivalent

Algorithm 9 Chase

Input: Φ : set of dependencies, \mathcal{W} : *WSD*

Output: a *WSD* satisfying Φ

```

1: for all  $d \in \Phi$  do
2:   if  $d = A_1, \dots, A_m \rightarrow A_0$  then //d is a fd
3:     for all  $s, t \in R : \{s, t\} \not\subseteq d$  do
4:       Let  $C_{j_i}, C_{k_i}$  be the component of  $s.A_i, t.A_i$ 
5:        $0 \leq i \leq m$ , respectively;
6:       replace  $C_{j_i}, C_{k_i}, 0 \leq i \leq m$ 
7:       by their product  $C$ ;
8:        $C := \sigma_d(C)$ ;
9:       if  $C = \emptyset$  then return  $\emptyset$ ;
10:      end if
11:    end for
12:   else if  $d = \phi_1 \wedge \dots \wedge \phi_m \rightarrow \phi_0$  then //d is a egd
13:     for all  $t \in R : \{t\} \not\subseteq d$  do
14:       Let  $C_i$  be the component of
15:        $t.A_i, 0 \leq i \leq m$ ;
16:       replace  $C_0, \dots, C_m$  by their product  $C$ ;
17:        $C := \sigma_d(C)$ ;
18:       if  $C = \emptyset$  then return  $\emptyset$ ;
19:     end if
20:   end for
21: end if
22: end for
23: return  $\mathcal{W}$ ;

```

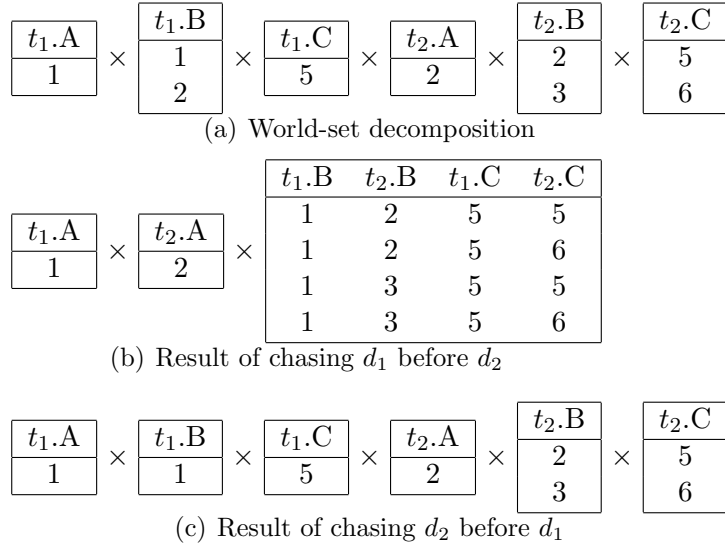


Figure 7.2: Impact of order on chasing

with respect to the set of possible worlds they represent. Indeed, the WSD in Figure 7.2 (b) can be reduced to the one in Figure 7.2 (c) using the simplification rules from Section 5.9.

The following theorems prove the correctness of the Chase algorithm.

Theorem 7.0.5 *Algorithm 9 terminates on all inputs.*

Theorem 7.0.6 (Correctness) *For a WSD \mathcal{W} and a set of dependencies Φ , Algorithm 9 computes a WSD \mathcal{W}' s.t. $\text{rep}(\mathcal{W}') \subseteq \text{rep}(\mathcal{W})$ and for each $\mathcal{A} \in \text{rep}(\mathcal{W})$,*

$$\mathcal{A} \in \text{rep}(\mathcal{W}') \Leftrightarrow \mathcal{A} \models \Phi.$$

Chapter 8

Experimental Evaluation

We evaluated experimentally the uniform world-set decompositions with template relations (UWSDTs), representing large census data with noise, by addressing the following questions: What is the space overhead of the additional relations used to represent incomplete information? Is the performance of query evaluation on UWSDTs comparable to the simplest case of querying one of their worlds? How big are UWSDTs after chasing a set of real-life dependencies on them?

The experiments of this section show that UWSDTs behave very well in practice. We found that the size of UWSDTs obtained as query answers or as result of chasing remains close to that of a single world. Furthermore, the processing time for queries is also comparable to processing one world. The explanation for this is that in practice there are rather few differences between the worlds represented by a UWSDT. This keeps the mapping and component relations relatively small and the lion's share of the processing time is taken by the templates.

Experimental Setting. The experiments are conducted on a 3GHz/2GB Pentium machine running Linux 2.6.8 and PostgreSQL 8.0. The queries are applied via the psql interface of PostgreSQL and the Chase via JDBC.

Datasets. The IPUMS 5% census data (Integrated Public Use Microdata Series, 1990) [25] used for the experiments is the publicly available 5% extract from the 1990 US census, amounting to 12491667 records (approx. 12.5 million). Each record contains 50 columns. The size of the relation stored in PostgreSQL is 3 GB. We also use excerpts representing the first 0.1%, 1% and 10% of the data.

We introduce noise by replacing some values with or-sets. We use different probabilities for the creation of an or-set: 0.005%, 0.01%, 0.05%. The selected probabilities reflect our assumptions about the data, namely that only a small percentage of the fields contain errors. The size of each or-set is chosen as

a random number in the range $[2, \min(8, size)]$, where $size$ is the size of the domain of the respective column (with a measured average of 3.5 values per or-set). Thus in one test scenario we have far more than 2^{312374} possible worlds, see Figure 8.3.

To obtain interesting cases for the chasing procedure, we cluster the or-sets by tuple. To do that we first select at random the tuples that should contain or-sets, and for each selected tuple we choose (again at random) between 1 and 4 columns to introduce or-sets to.

1	CITIZEN	= 0	⇒	IMMIGR	= 0
2	FEB55	= 1	⇒	MILITARY	! = 4
3	KOREAN	= 1	⇒	MILITARY	! = 4
4	VIETNAM	= 1	⇒	MILITARY	! = 4
5	WWII	= 1	⇒	MILITARY	! = 4
6	MARITAL	= 0	⇒	RSPOUSE	! = 6
7	MARITAL	= 0	⇒	RSPOUSE	! = 5
8	LANG1	= 2	⇒	ENGLISH	! = 4
9	RPOB	= 52	⇒	CITIZEN	! = 0
10	SCHOOL	= 0	⇒	KOREAN	! = 1
11	SCHOOL	= 0	⇒	FEB55	! = 1
12	SCHOOL	= 0	⇒	WWII	! = 1

Figure 8.1: Dependencies for cleaning census data

Data Cleaning. We enhance our Chase algorithm of Section 7 in several ways. For a given dependency we retrieve all inconsistencies at once instead of doing that tuplewise, and we perform as many updates as possible at the same time, instead of having one update per inconsistent tuple. This is in the spirit of the semi-naive algorithm for datalog evaluation [1].

The Chase uses the 12 equality-generating dependencies from Figure 8.1. These represent real-life constraints on the data. For example the first dependency states that citizens born in the USA are not immigrants, and dependencies two to five require that citizens who served in various wars have done their military service.

Figure 8.3 shows the effect of chasing our dependencies on the 12.5 million tuples and varying placeholder density. The number of components being merged grows linearly with the increase of placeholder density. A linear increase is witnessed also by the chasing time when the number of tuples is also varied, cf. Figure 8.2.

Queries. Six queries were chosen to show the behavior of relational operators combinations under varying selectivities (cf. Figure 8.4). Query Q_1

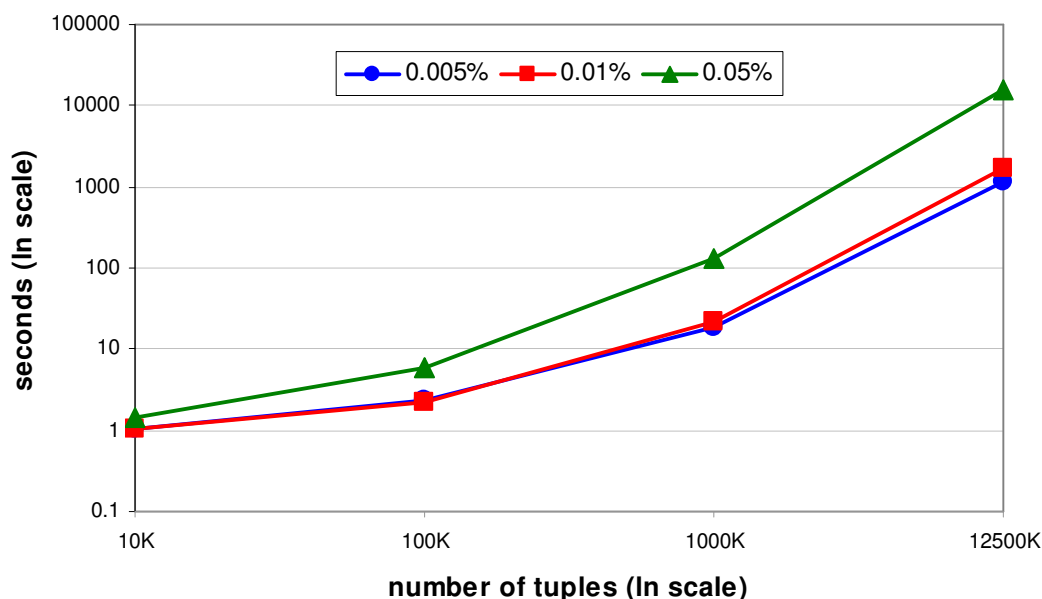


Figure 8.2: Time for chasing the dependencies of Figure 8.1 on UWSDTs of various sizes and densities.

returns the entries of US citizens with PhD degree. The less selective query Q_2 returns the place of birth of US citizens not born in the US that do not speak English well. Query Q_3 retrieves the entries of widows that have more than three children and that live in the state in which they were born. Query Q_4 finds the entries of 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, etc.). Finally, query Q_6 retrieves the places of birth and work of persons speaking English well.

Figure 8.3 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 and of components with at least two placeholders, the number of tuples in the component relation, and the size of the template relation. 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.

Figures 8.5 to 8.10 show that all six queries admit efficient and scalable evaluation on UWSDTs of different sizes (10K, 100K, 1M, and 12.5M) and placeholder densities (0, 0.005%, 0.01%, and 0.05%). Although the evaluation of join conditions on decompositions can require theoretically expo-

Density	0.005%	0.01%	0.05%
Before chasing			
#components	31154	62619	312374
After chasing			
#components	30859	62003	309545
#comp>1	284	588	2725
comp size	108580	211115	1089238
template size	12.5M	12.5M	12.5M
After Q_1			
#components	648	1352	6523
#comp>1	3	9	31
#comp size	1688	3683	17737
template size	46600	46780	48223
After Q_2			
#components	25	52	2059
#comp>1	0	0	25
comp size	121	234	3830
template size	82990	83045	84143
After Q_3			
#components	40	65	384
#comp>1	0	0	0
comp size	114	144	968
template size	17917	17935	18176
After Q_4			
#components	1544	3043	15550
#comp>1	13	28	130
comp size	4762	9435	47432
template size	402327	402543	403999
After Q_5			
#components	6	8	43
#comp>1	0	2	13
comp size	18	830	2014
template size	17941	18923	19123
After Q_6			
#components	81	170	6837
#comp>1	0	0	0
comp size	451	850	13096
template size	229574	230076	237882

Figure 8.3: UWSDTs characteristics before chasing and after chasing and querying for 12.5M tuples

$$\begin{aligned}
 Q_1 &:= \sigma_{\text{YEARSCH}=17 \wedge \text{CITIZEN}=0}(R) \\
 Q_2 &:= \pi_{\text{POWSTATE}, \text{CITIZEN}, \text{IMMIGR}}(\sigma_{\text{CITIZEN} < > 0 \wedge \text{ENGLISH} > 3}(R)) \\
 Q_3 &:= \pi_{\text{POWSTATE}, \text{MARITAL}, \text{FERTIL}}(\sigma_{\text{POWSTATE}=\text{POB}} \\
 &\quad (\sigma_{\text{FERTIL} > 4 \wedge \text{MARITAL}=1}(R))) \\
 Q_4 &:= \sigma_{\text{FERTIL}=1 \wedge (\text{RSPOUSE}=1 \vee \text{RSPOUSE}=2)}(R) \\
 Q_5 &:= \delta_{\text{POWSTATE} \rightarrow P_1}(\sigma_{\text{POWSTATE} > 50}(Q_2)) \bowtie_{P_1=P_2} \\
 &\quad \delta_{\text{POWSTATE} \rightarrow P_2}(\sigma_{\text{POWSTATE} > 50}(Q_3)) \\
 Q_6 &:= \pi_{\text{POWSTATE}, \text{POB}}(\sigma_{\text{ENGLISH}=3}(R))
 \end{aligned}$$

Figure 8.4: Queries on IPUMS census data

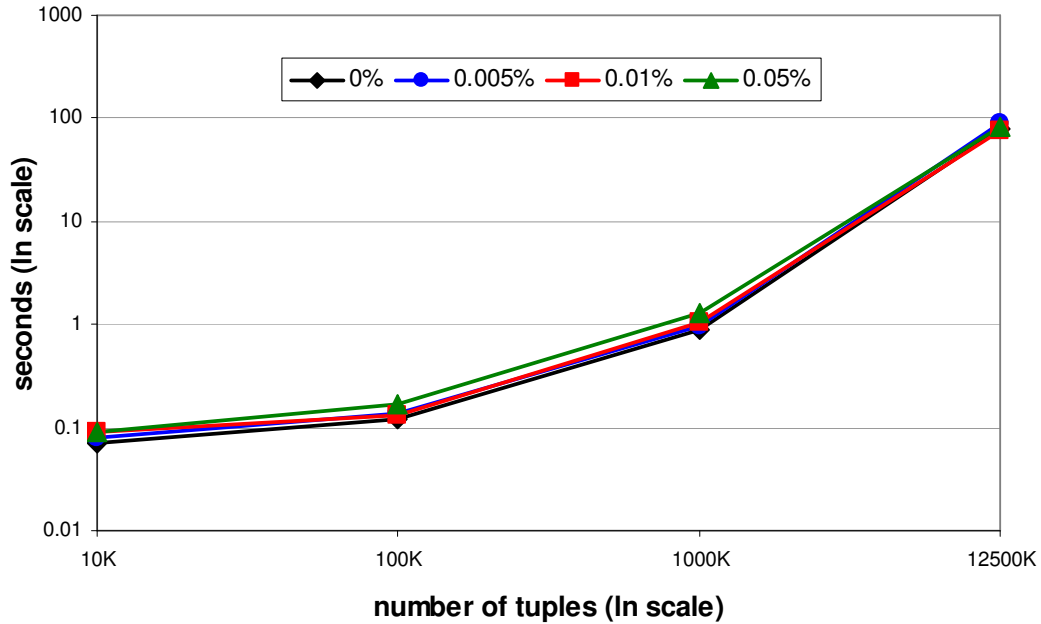


Figure 8.5: Evaluation time for Query Q_1 .

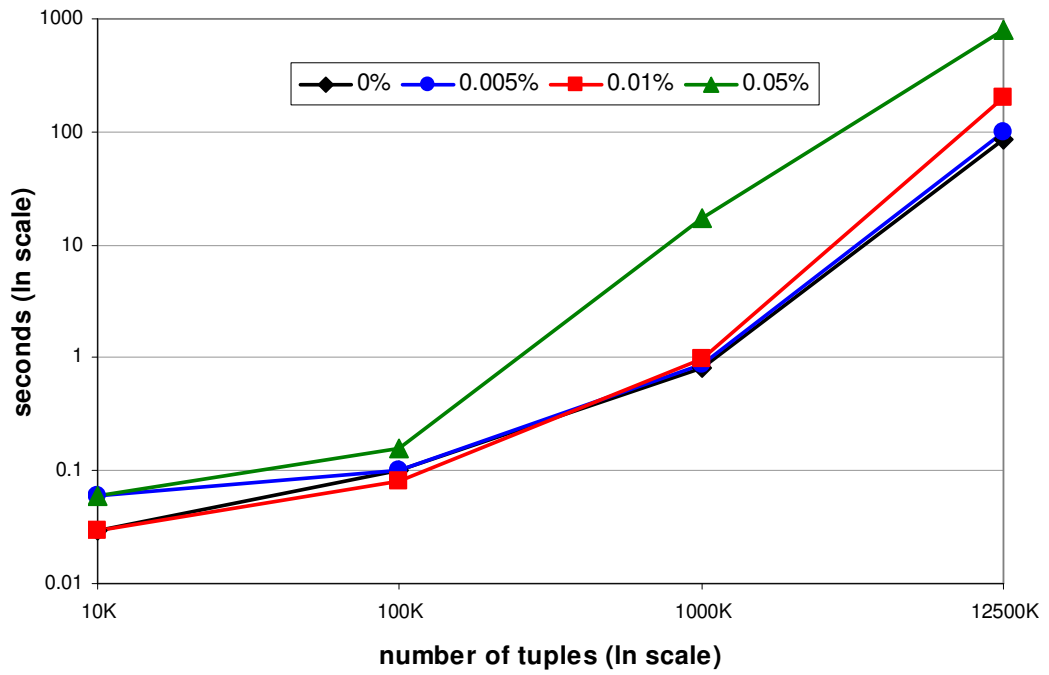


Figure 8.6: Evaluation time for Query Q_2 .

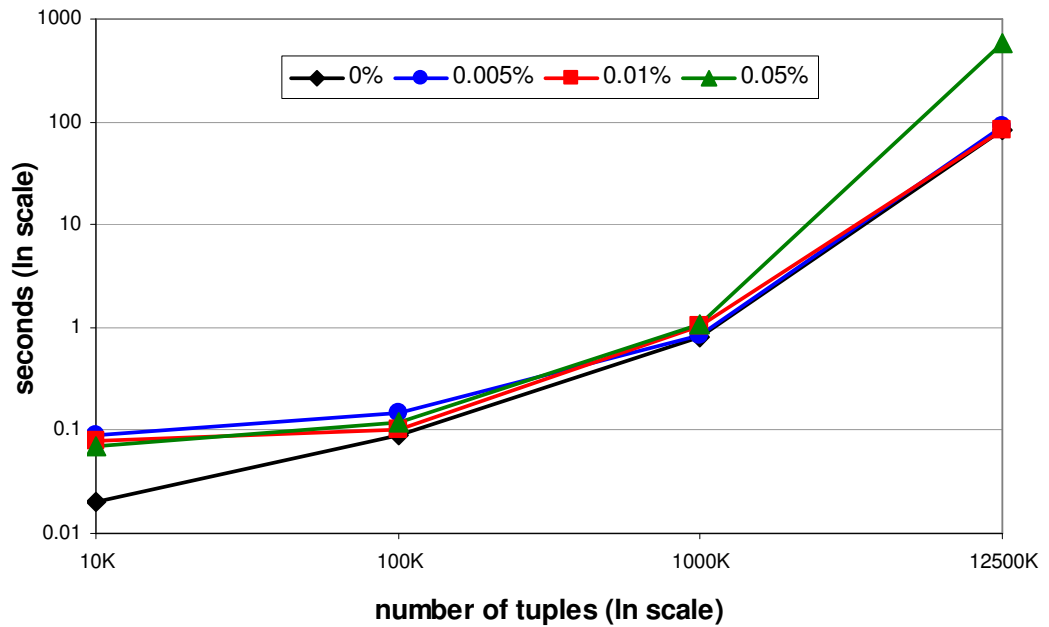


Figure 8.7: Evaluation time for Query Q_3 .

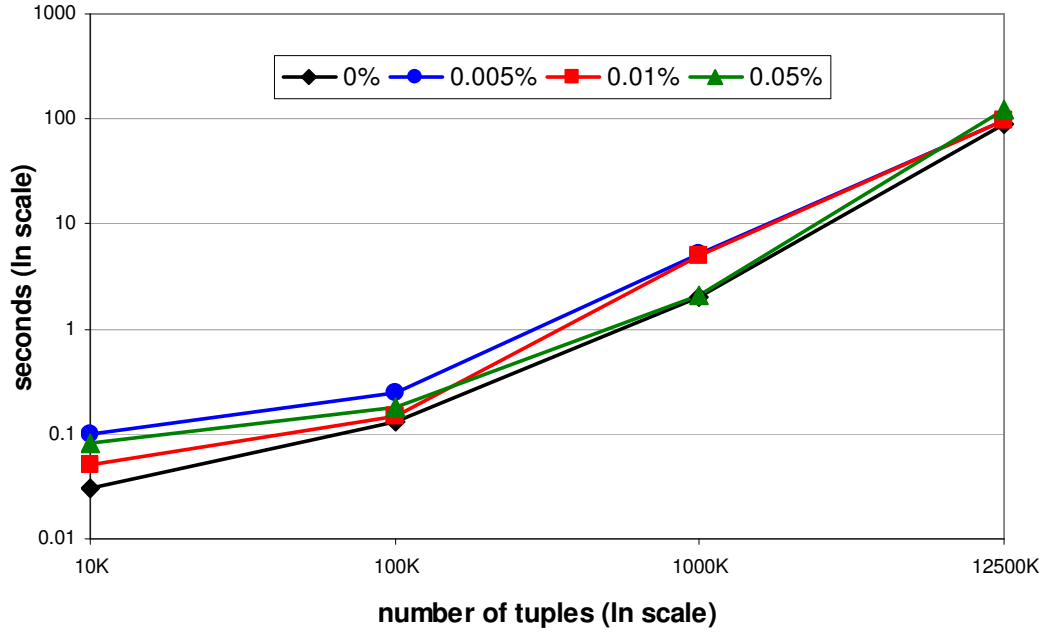


Figure 8.8: Evaluation time for Query Q_4 .

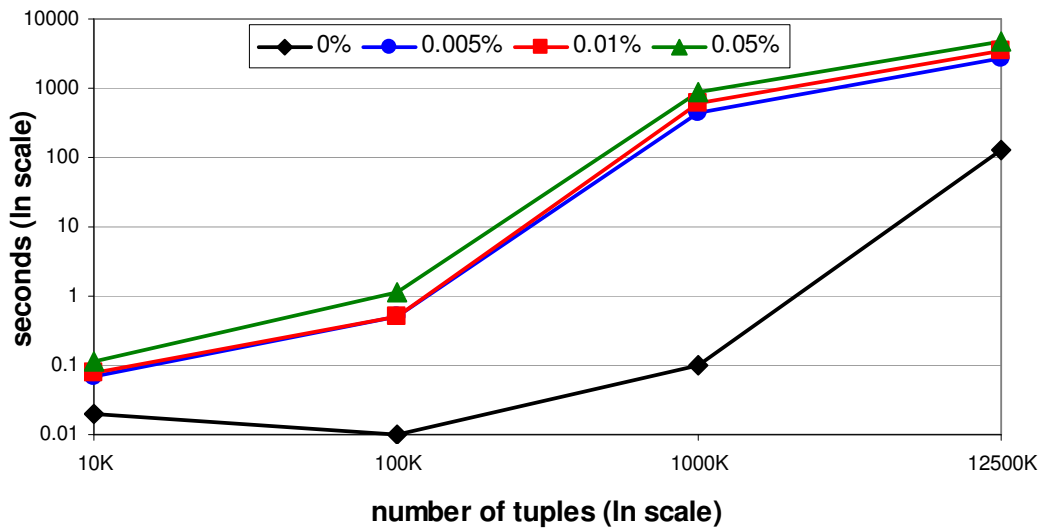
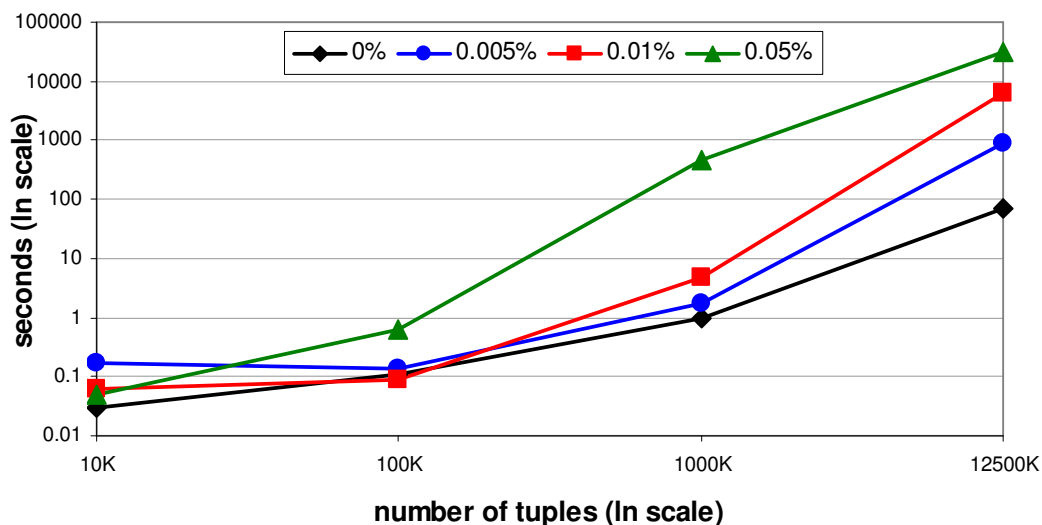


Figure 8.9: Evaluation time for Query Q_5 .

Figure 8.10: Evaluation time for Query Q_6 .

nential evaluation time, our experiments suggest that they behave well in practical cases, as illustrated in Figures 8.7 and 8.9 for queries Q_3 and Q_5 .

The experiments answer positively also our initial question relating the performance of query evaluation on UWSDTs to the one-world case. All but two of our queries perform very good and close to the one-world case (density zero in the diagrams of Figure 8.5 to 8.10). Queries Q_5 and Q_6 are the most expensive ones and exhibit an important time increase when compared to the one-world case. This can be explained by the join operation of Q_5 and the unselective condition of Q_6 .

Chapter 9

Conclusions and Future Work

In this thesis we presented a solution for representing and querying finite sets of worlds and we showed that our model is a strong representation system for any relational query language.

While first experiments suggest that WSDs are a very promising framework for representing and managing incomplete information, we are currently working on evaluating WSDs in applications other than survey data. We would like to explore cases where the data is much more interdependent and the dependencies involve multiple rows and columns of the relations, or scenarios where data contains a higher percentage of errors. It would be interesting to find a threshold in terms of error and interdependency ratio, above which world-set decompositions become impractical and inconvenient to use.

Our main contribution regarding data cleaning was to embed it into our framework and to produce realistic world-sets beyond or-set relations for our experiments. We plan to embark on deeper work on data cleaning within our framework.

We would also like to investigate the problem of decomposing a world set when all possible worlds are given, rather than producing the decomposition by chasing dependencies. An optimal solution might be intractable. Nevertheless finding a good algorithm would be of interest.

We did not address the problem of making *updates* to WSDs in this thesis, even though it is easy to handle in the case that changes made are the same in all worlds. (For instance, adding a tuple to each world of a UWSDT just means to add the tuple once to the template relation.) However, updating world-sets with incomplete information – even defining meaningful notions of this – is an interesting problem for future research.

We plan to generalize the framework of WSDs to be able to support also infinite sets of possible worlds by taking an approach that integrates WSDs

with c-tables. Our WSDs correspond to a certain CNF-like *normal form* for the condition formulae in c-tables. In the future, we would like to elaborate on this to define syntactic normal forms for c-tables which are at once powerful as representation systems (for finite as well as infinite world-sets) and allow for scalable representation, data cleaning, and query processing.

Schema and codes for IPUMS data

AAUGMENT: Augmented Pers.

0 No
1 Yes

ABIRTHPL: Place of Birth

0 No
1 Yes

AGE: Age

00 Less Than 1 Year
90 90 or More Yrs. Old

ANCSTRY1: Ancestry First Entry

999 Not Reported

ANCSTRY2: Ancestry Second Entry

000 No Secondary Ancestry
999 Not Reported

AVAIL: Available for Work

0 N/a Less Than 16 Yrs./at Work/not Lookin
1 No, Already Has a Job
2 No, Temply. Ill
3 No, Other Reasons in School, Etc.
4 Yes, Could Have Taken a Job

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

CITIZEN: Citizenship

- 0 Born in the U.S.
- 1 Born in Puerto Rico, Guam, and Outlying
- 2 Born Abroad of American Parents
- 3 U.S. Citizen by Naturalization
- 4 Not a Citizen of the U.s

CLASS: Class of Worker

- 0 N/a Less Than 16 Yrs. Old/unemp. Who Nev
- 1 Emp. of a Private for Profit Company or
- 2 Emp. of a Private Not for Profit, Tax Ex
- 3 Local Gov. Emp. City, County, Etc.
- 4 State Gov. Emp.
- 5 Fed. Gov. Emp.
- 6 Self Emp. in Own Not Incorp.d Business,
- 7 Self Emp. in Own Incorp.d Business, Prof
- 8 Working Without Pay in Fam. Bus. or Farm
- 9 Unemp., Last Worked in 1984 or Earlier

ENGLISH: Ability to Speak English

- 0 N/a Less Than 5 Yrs. Old/speaks Only Eng
- 1 Very Well
- 2 Well
- 3 Not Well
- 4 Not At All

FEB55: Served February 1955 July 1964

- 0 Did Not Serve This Per./less Than 16 Yr
- 1 Served This Per.

FERTIL: No. of Chld. Ever Born

- 00 N/a Less Than 15 Yrs./male
- 01 No Chld.
- 02 1 Child
- 03 2 Chld.
- 04 3 Chld.
- 05 4 Chld.

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

06 5 Chld.
07 6 Chld.
08 7 Chld.
09 8 Chld.
10 9 Chld.
11 10 Chld.
12 11 Chld.
13 12 or More Chld.

HISPANIC: Detailed Hispanic Origin Code

000 Not Hispanic 006 199
001 Mexican, Mex Am 210 220
002 Puerto Rican 261 270
003 Cuban 271 274
004 Other Hispanic 200 209, 250 260, 290 401

IMMIGR: Yr. of Entry

00 Born in the U.S.
01 1987 to 1990
02 1985 to 1986
03 1982 to 1984
04 1980 or 1981
05 1975 to 1979
06 1970 to 1974
07 1965 to 1969
08 1960 to 1964
09 1950 to 1959
10 Before 1950

INCOME1: Wages or Salary Inc. in 1989

000000 N/a Less Than 16 Yrs. Old/none
140000 Topcode
140001 140001 or More State Median of Topcoded

KOREAN: Served Korean Conflict June 1950 January

0 Did Not Serve This Per./less Than 16 Yr
1 Served This Per.

LANG1: Language Other Than English At Home

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

- 0 N/a Less Than 5 Yrs. Old
- 1 Yes, Speaks Another Language
- 2 No, Speaks Only English

MARITAL: Marital Stat.

- 0 Now Married, Except Separated
- 1 Widowed
- 2 Divorced
- 3 Separated
- 4 Never Married or Under 15 Yrs. Old

MAY75880: Served May 1975 to August 1980

- 0 Did Not Serve This Per./less Than 16 Yr
- 1 Served This Per.

MEANS: Means of Transportation to Work

- 00 N/a Not a Worker Not in the Labor Force,
- 01 Car, Truck, or Van
- 02 Bus or Trolley Bus
- 03 Streetcar or Trolley Car
- 04 Subway or Elevated
- 05 Railroad
- 06 Ferryboat
- 07 Taxicab
- 08 Motorcycle
- 09 Bicycle
- 10 Walked
- 11 Worked At Home
- 12 Other Method

MIGPUMA: Migration Puma State Dependent

- 00000 N/a Pers. Less Than 5 Yrs. Old/lived in
- 99900 Abroad

MIGSTATE: Migration State or Foreign Country Code

- 00 N/a Pers. Less Than 5 Yrs. Old/lived in
- 01 Alabama

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

02 Alaska
04 Arizona
05 Arkansas
06 California
08 Colorado
09 Connecticut
10 Delaware
11 District of Columbia
12 Florida
13 Georgia
15 Hawaii
16 Idaho
17 Illinois
18 Indiana
19 Iowa
20 Kansas
21 Kentucky
22 Louisiana
23 Maine
24 Maryland
25 Massachusetts
26 Michigan
27 Minnesota
28 Mississippi
29 Missouri
30 Montana
31 Nebraska
32 Nevada
33 New Hampshire
34 New Jersey
35 New Mexico
36 New York
37 North Carolina
38 North Dakota
39 Ohio
40 Oklahoma
41 Oregon
42 Pennsylvania
44 Rhode Island
45 South Carolina
46 South Dakota
47 Tennessee
48 Texas

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

49 Utah
50 Vermont
51 Virginia
53 Washington
54 West Virginia
55 Wisconsin
56 Wyoming
72 Puerto Rico
98 Other Abroad in 1985
99 State Not Identified B Sample

MILITARY: Military Srvc.

0 N/a Less Than 16 Yrs. Old
1 Yes, Now on Active Duty
2 Yes, on Active Duty in Past, But Not Now
3 Yes, Srvc. in Reserves or Nat. Guard Onl
4 No Srvc.

OCCUP: Occupation

000 N/a Less Than 16 Yrs. Old/unemp. Who Nev

OTHRSERV: Served Any Other Time

0 Did Not Serve This Per./less Than 16 Yr
1 Served This Per.

POB: Place of Birth

001 Alabama
002 Alaska
004 Arizona
005 Arkansas
006 California
008 Colorado
009 Connecticut
010 Delaware
011 District of Columbia
012 Florida
013 Georgia
015 Hawaii
016 Idaho

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

017 Illinois
018 Indiana
019 Iowa
020 Kansas
021 Kentucky
022 Louisiana
023 Maine
024 Maryland
025 Massachusetts
026 Michigan
027 Minnesota
028 Mississippi
029 Missouri
030 Montana
031 Nebraska
032 Nevada
033 New Hampshire
034 New Jersey
035 New Mexico
036 New York
037 North Carolina
038 North Dakota
039 Ohio
040 Oklahoma
041 Oregon
042 Pennsylvania
044 Rhode Island
045 South Carolina
046 South Dakota
047 Tennessee
048 Texas
049 Utah
050 Vermont
051 Virginia
053 Washington
054 West Virginia
055 Wisconsin
056 Wyoming
060 American Samoa
066 Guam
067 Johnston Atoll
069 Northern Mariana Islands
071 Midway Islands

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

072 Puerto Rico
076 Navassa Island
078 U.S. Virgin Islands
079 Wake Island
081 Baker Island
084 Howland Island
086 Jarvis Island
089 Kingman Reef
095 Palmyra Atoll
096 U.S. Territory, Not Specified
100 Albania
101 Andorra
102 Austria
103 Belgium
104 Bulgaria
105 Czechoslovakia
106 Denmark
107 Faroe Islands
108 Finland
109 France
110 Germany, Not Specified
111 West Germany
112 West Berlin
113 East Berlin
114 East Germany
115 Gibraltar
116 Greece
117 Hungary
118 Iceland
119 Ireland
120 Italy
121 Jan Mayen
122 Liechtenstein
123 Luxembourg
124 Malta
125 Monaco
126 Netherlands
127 Norway
128 Poland
129 Portugal
130 Azores Islands
131 Madeira Islands
132 Romania

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

133 San Marino
134 Spain
135 Svalbard
136 Sweden
137 Switzerland
138 United Kingdom, Not Specified
139 England
140 Scotland
141 Wales
142 Northern Ireland
143 Guernsey
144 Jersey
145 Isle of Man
146 Vatican City
147 Yugoslavia
148 Europe, Not Specified
149 Central Europe, Not Specified
150 Eastern Europe, Not Specified
151 Lapland, Not Specified
152 Northern Europe, Not Specified
153 Southern Europe, Not Specified
154 Western Europe, Not Specified
180 Union of Soviet Soc.ist Repub.s U.S.
181 Baltic States, Not Specified
182 Estonia
183 Latvia
184 Lithuania
200 Afghanistan
201 Bahrain
202 Bangladesh
203 Bhutan
204 Brunei
205 Burma
206 Cambodia
207 China
208 Cyprus
209 Hong Kong
210 India
211 Indonesia
212 Iran
213 Iraq
214 Israel
215 Japan

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

216 Jordan
217 Korea, Not Specified
218 South Korea
219 North Korea
220 Kuwait
221 Laos
222 Lebanon
223 Macau
224 Malaysia
225 Maldives
226 Mongolia
227 Nepal
228 Oman
229 Pakistan
230 Paracel Islands
231 Philippines
232 Qatar
233 Saudi Arabia
234 Singapore
235 Spratley Islands
236 Sri Lanka
237 Syria
238 Taiwan
239 Thailand
240 Turkey
241 United Arab Emirates
242 Vietnam
243 Yemen, Peoples Democratic Repub.
244 Yemen Arab Repub.
245 Asia, Not Specified
246 Asia Minor, Not Specified
247 East Asia, Not Specified
248 Gaza Strip
249 Indochina, Not Specified
250 Iraq Saudi Arabia Neutral Zone
251 Mesopotamia, Not Specified
252 Middle East, Not Specified
253 Palestine, Not Specified
254 Persian Gulf States, Not Specified
255 Southeast Asia, Not Specified
256 West Bank
300 Bermuda
301 Canada

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

302 Greenland
303 St. Pierre and Miquelon
304 North America, Not Specified
310 Belize
311 Costa Rica
312 El Salvador
313 Guatemala
314 Honduras
315 Mexico
316 Nicaragua
317 Panama
318 Central America, Not Specified
330 Anguilla
331 Antigua and Barbuda
332 Aruba
333 Bahamas
334 Barbados
335 British Virgin Islands
336 Cayman Islands
337 Cuba
338 Dominica
339 Dominican Repub.
340 Grenada
341 Guadeloupe
342 Haiti
343 Jamaica
344 Martinique
345 Montserrat
346 Netherlands Antilles
347 St. Barthelemy
348 St. Kitts Nevis
349 St. Lucia
350 St. Vincent and the Grenadines
351 Trinidad and Tobago
352 Turks and Caicos Islands
353 Caribbean, Not Specified
354 Antilles, Not Specified
355 British West Indies, Not Specified
356 Latin America, Not Specified
357 Leeward Islands, Not Specified
358 West Indies, Not Specified
359 Windward Islands, Not Specified
375 Argentina

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

376 Bolivia
377 Brazil
378 Chile
379 Colombia
380 Ecuador
381 Falkland Islands
382 French Guiana
383 Guyana
384 Paraguay
385 Peru
386 Suriname
387 Uruguay
388 Venezuela
389 South America, Not Specified
400 Algeria
401 Angola
402 Bassas Da India
403 Benin
404 Botswana
405 British Indian Ocean Territory
406 Burkina Faso
407 Burundi
408 Cameroon
409 Cape Verde
410 Central African Repub.
411 Chad
412 Comoros
413 Congo
414 Djibouti
415 Egypt
416 Equatorial Guinea
417 Ethiopia
418 Europa Island
419 Gabon
420 Gambia
421 Ghana
422 Glorioso Islands
423 Guinea
424 Guinea Bissau
425 Ivory Coast
426 Juan De Nova Island
427 Kenya
428 Lesotho

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

429 Liberia
430 Libya
431 Madagascar
432 Malawi
433 Mali
434 Mauritania
435 Mayotte
436 Morocco
437 Mozambique
438 Namibia
439 Niger
440 Nigeria
441 Reunion
442 Rwanda
443 Sao Tome and Principe
444 Senegal
445 Mauritius
446 Seychelles
447 Sierra Leone
448 Somalia
449 South Africa
450 St. Helena
451 Sudan
452 Swaziland
453 Tanzania
454 Togo
455 Tromelin Island
456 Tunisia
457 Uganda
458 Western Sahara
459 Zaire
460 Zambia
461 Zimbabwe
462 Africa, Not Specified
463 Central Africa, Not Specified
464 Eastern Africa, Not Specified
465 Equatorial Africa, Not Specified
466 French Equatorial Africa, Not Specified
467 French West Africa, Not Specified
468 North Africa, Not Specified
469 Western Africa, Not Specified
470 Southern Africa, Not Specified
500 Ashmore and Cartier Islands

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

501 Australia
502 Christmas Island, Indian Ocean
503 Clipperton Island
504 Cocos Islands
505 Cook Islands
506 Coral Sea Islands
507 Fiji
508 French Polynesia
509 Kiribati
510 Marshall Islands
511 Micronesia
512 Nauru
513 New Caledonia
514 New Zealand
515 Niue
516 Norfolk Island
517 Palau
518 Papua New Guinea
519 Pitcairn Islands
520 Solomon Islands
521 Tokelau
522 Tonga
523 Tuvalu
524 Vanuatu
525 Wallis and Futuna Islands
526 Western Samoa
527 Oceania, Not Specified
528 Polynesia, Not Specified
529 Melanesia, Not Specified
550 Antarctica
551 Bouvet Island
552 French Southern and Antarctic Lands
553 Heard and McDonald Islands
554 At Sea
555 Abroad, Not Specified

POWSTATE: Place of Work State

With the following Ranges:

00 N/a Not a Worker Not in the Labor Force,
01 Alabama
02 Alaska
04 Arizona

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

05 Arkansas
06 California
08 Colorado
09 Connecticut
10 Delaware
11 District of Columbia
12 Florida
13 Georgia
15 Hawaii
16 Idaho
17 Illinois
18 Indiana
19 Iowa
20 Kansas
21 Kentucky
22 Louisiana
23 Maine
24 Maryland
25 Massachusetts
26 Michigan
27 Minnesota
28 Mississippi
29 Missouri
30 Montana
31 Nebraska
32 Nevada
33 New Hampshire
34 New Jersey
35 New Mexico
36 New York
37 North Carolina
38 North Dakota
39 Ohio
40 Oklahoma
41 Oregon
42 Pennsylvania
44 Rhode Island
45 South Carolina
46 South Dakota
47 Tennessee
48 Texas
49 Utah
50 Vermont

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

51 Virginia
53 Washington
54 West Virginia
55 Wisconsin
56 Wyoming
98 Abroad
99 State Not Identified

RACE: Recoded Detailed Race Code

001 White 800 869, 971
002 Black 870 934, 972
004 Eskimo 935 940, 974
005 Aleut 941 970, 975
006 Chinese, Except Taiwanese 605, 976
007 Taiwanese 606, 607
008 Filipino 608, 977
009 Japanese 611, 981
010 Asian Indian 600, 982
011 Korean 612, 979
012 Vietnamese 619, 980
013 Cambodian 604
014 Hmong 609
015 Laotian 613
016 Thai 618
017 Bangladeshi 601
018 Burmese 603
019 Indonesian 610
020 Malayan 614
021 Okinawan 615
022 Pakistani 616
023 Sri Lankan 617
024 All Other Asian 602, 620 652, 985
025 Hawaiian 653, 654, 978
026 Samoan 655, 983
027 Tahitian 656
028 Tongan 657
029 Other Polynesian 658, 659
030 Guamanian 660, 984
031 Northern Mariana Islander 661, 671, 673
032 Palauan 663
033 Other Micronesian 662, 664 670, 672, 674
034 Fijian 676

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

035 Other Melanesian 677 680
036 Pacific Islander, Not Specified 681 699
037 Other Race 700 799, 986 999
301 Alaskan Athabaskan 000, 001, 008, 009, 0
302 Apache 255 264
303 Blackfoot 360
304 Cherokee 416 422, 555 557, 562
305 Cheyenne 361 363
306 Chickasaw 436
307 Chippewa 330 353, 355, 544
308 Choctaw 226, 228, 404, 434, 520, 559
309 Comanche 325, 523
310 Creek 423, 425, 426, 429 432, 449, 540,
311 Crow 322
312 Iroquois 405 415
313 Kiowa 276, 522
314 Lumbee 464
315 Navajo 275
316 Osage 320
317 Paiute 175 192, 542
318 Pima 217
319 Potawatomi 367 374
320 Pueblo 229 254, 506, 573
321 Seminole 428, 438 443
322 Shoshone 195 206, 494, 518
323 Sioux 282 312, 326, 327
324 Tlingit 017
325 Tohono Oodham 218 222
326 All Other Tribes 002 007, 010 013, 015,
327 Tribe Not Specified 548, 549, 576 598 Tr

RAGECHLD: Presence and Age of Own Chld.

0 N/a Male
1 With Own Chld. Under 6 Yrs. Only
2 With Own Chld. 6 to 17 Yrs. Only
3 With Own Chld. Under 6 Yrs. and 6 to 17
4 No Own Chld. .incl. Females Under 16 Yrs

RELAT1: Rel. or Not Related or Grp. Qtrs.

00 Hshldr.
01 Husband/wife

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

02 Son/daughter
03 Stepson/stepdaughter
04 Brother/sister
05 Father/mother
06 Grandchild
07 Other Rel.
08 Roomer/boarder/foster Child
09 Housemate/roommate
10 Unmarried Partner
11 Other Nonrel.
12 Instit. Person
13 Other Pers. in Grp. Qtrs.

RELAT2: Detailed Rel. Other Rel.

0 N/a Gq/not Other Rel.
1 Son in Law/daughter in Law
2 Father in Law/mother in Law
3 Brother in Law/sister in Law
4 Nephew/niece
5 Grandparent
6 Uncle/aunt
7 Cousin
8 Other Related by Blood or Marriage
9 Other Rel.

REMPAR: Employment Stat. of Parents

000 N/a Not Own Child of Hshldr., and Not Ch
111 Both Parents At Work 35 or More Hrs.
112 Father Only At Work 35 or More Hrs.
113 Mother Only At Work 35 or More Hrs.
114 Neither Parent At Work 35 or More Hrs.
121 Father At Work 35 or More Hrs.
122 Father Not At Work 35 or More Hrs.
133 Mother At Work 35 or More Hrs.
134 Mother Not At Work 35 or More Hrs.
141 Neither Parent in Labor Force
211 Father At Work 35 or More Hrs.
212 Father Not At Work 35 or More Hrs.
213 Father Not in Labor Force
221 Mother At Work 35 or More Hrs.
222 Mother Not At Work 35 or More Hrs.

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

223 Mother Not in Labor Force

RIDERS: Vehicle Occupancy

0 N/a Not a Worker or Worker Whose Means o
1 Drove Alone
2 2 People
3 3 People
4 4 People
5 5 People
6 6 People
7 7 to 9 People
8 10 or More People

RLABOR: Employment Stat. Recode

0 N/a Less Than 16 Yrs. Old
1 Civilian Emp., At Work
2 Civilian Emp., With a Job But Not At Wor
3 Unemp.
4 Armed Forces, At Work
5 Armed Forces, With a Job But Not At Work
6 Not in Labor Force

ROWNCHLD: Own Child

0 Not Own Child
1 Own Child

RPOB: Place of Birth Recode

10 Born in State of Res.
21 Northeast
22 Midwest
23 South
24 West
31 Puerto Rico
32 American Samoa
33 Guam
34 Northern Marianas
35 Us Virgin Islands
36 Elsewhere
40 Born Abroad of American Parents

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

51 Naturalized Citizen
52 Not a Citizen

RSPOUSE: Married, Spouse Present/spouse Absent

0 N/a Less Than 15 Yrs. Old
1 Now Married, Spouse Present
2 Now Married, Spouse Absent
3 Widowed
4 Divorced
5 Separated
6 Never Married

RVETSERV: Veteran Per. of Srvc.

00 N/a Less Than 16 Yrs. Old, No Active Dut
01 September 1980 or Later Only
02 May 1975 to August 1980 Only
03 May 1975 to August 1980 and September 19
04 Vietnam Era, No Korean Conflict, No Wwii
05 Vietnam Era and Korean Conflict, No Wwii
06 Vietnam Era and Korean Conflict and Wwii
07 February 1955 to July 1964 Only
08 Korean Conflict, No Vietnam Era, No Wwii
09 Korean Conflict and Wwii, No Vietnam Era
10 Wwii, No Korean Conflict, No Vietnam Era
11 Other Srvc.

SCHOOL: School Enrollment

0 N/a Less Than 3 Yrs. Old
1 Not Attending School
2 Yes, Pub. School, Pub. Coll.
3 Yes, Private School, Private Coll.

SEPT80: Served September 1980 or Later

0 Did Not Serve This Per./less Than 16 Yr
1 Served This Per.

SERIALNO: Hu/gq Pers. Serial No. Unique Within Sta

SEX: Sex

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

- 0 Male
- 1 Female

SUBFAM1: Subfam. Rel.

- 0 N/a Gq/not in a Subfam.
- 1 Husband/wife
- 2 Parent in a Parent/child Subfam.
- 3 Child in Subfam.

SUBFAM2: Subfam. Number

- 0 N/a Gq/not in a Subfam.
- 1 Husband/wife
- 2 Parent in a Parent/child Subfam.
- 3 Child in Subfam.

TMPABSNT: Temp. Absence From Work

- 0 N/a Less Than 16 Yrs. Old/at Work/did No
- 1 Yes, on Layoff
- 2 Yes, on Vacation, Temp. Illness, Labor D
- 3 No

TRAVTIME: Travel Time to Work

- 00 N/a Not a Worker or Worker Who Worked At
- 99 99 Minutes or More to Get to Work

VIETNAM: Served Vietnam Era August 1964 April 197

- 0 Did Not Serve This Per./less Than 16 Yr
- 1 Served This Per.

WWII: Served World War II September 1940 July

- 0 Did Not Serve This Per./less Than 16 Yr
- 1 Served This Per.

YEARSCH: Ed. Attainment

- 00 N/a Less Than 3 Yrs. Old

APPENDIX . SCHEMA AND CODES FOR IPUMS DATA

01 No School Completed
02 Nursery School
03 Kindergarten
04 1st, 2nd, 3rd, or 4th Grade
05 5th, 6th, 7th, or 8th Grade
06 9th Grade
07 10th Grade
08 11th Grade
09 12th Grade, No Diploma
10 High School Graduate, Diploma or Ged
11 Some Coll., But No Degree
12 Associate Degree in Coll., Occupational
13 Associate Degree in Coll., Academic Prog
14 Bachelors Degree
15 Masters Degree
16 Professional Degree
17 Doctorate Degree

YEARWRK: Yr. Last Worked

0 N/a Less Than 16 Yrs. Old
1 1990
2 1989
3 1988
4 1985 to 1987
5 1980 to 1984
6 1979 or Earlier
7 Never Worked

YRSSERV: Yrs. of Active Duty Military Srvc.

00 N/a Less Than 16 Yrs./no Active Duty Mil
01 1 Yr. or Less of Srvc.
50 50 or More Yrs. of Srvc.

Bibliography

- [1] Serge Abiteboul, Richard Hull, and Victor Vianu. “*Foundations of Databases*”. Addison-Wesley, 1995.
- [2] Serge Abiteboul, Paris Kanellakis, and Gösta Grahne. “On the Representation and Querying of Sets of Possible Worlds”. *Theoretical Computer Science*, 78(1):158–187, 1991.
- [3] Serge Abiteboul, Luc Segoufin, and Victor Vianu. “Representing and Querying XML with Incomplete Information”. In *PODS*, pages 150–161, 2001.
- [4] Alfred V. Aho, Catriel Beeri, and Jeffrey D. Ullman. “The Theory of Joins in Relational Databases”. *ACM Transactions on Database Systems*, 4(3):297–314, 1979.
- [5] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. “Consistent Query Answers in Inconsistent Databases”. In *PODS*, pages 68–79, 1999.
- [6] Jose A. Blakeley, Per-Ake Larson, and Frank Wm Tompa. “Efficiently Updating Materialized Views”. In *SIGMOD*, pages 61–71, 1986.
- [7] Philip Bohannon, Wenfei Fan, Michael Flaster, and Rajeev Rastogi. “A Cost-Based Model and Effective Heuristic for Value-Based Constraint Repairing”. In *SIGMOD*, pages 143–154, June 2005.
- [8] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. “Robust and Efficient Fuzzy Match for Online Data Cleaning”. In *SIGMOD*, pages 313–324, 2003.
- [9] E. F. Codd. “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM*, 13(6):377–387, 1970.
- [10] E. F. Codd. “Understanding Relations (Installment #7)”. *FDT - Bulletin of ACM SIGMOD*, 7(3):23–28, 1975.

BIBLIOGRAPHY

- [11] E. F. Codd. “Extending the Database Relational Model to Capture More Meaning”. *ACM Transactions on Database Systems*, 4(4):397–434, 1979.
- [12] Ariel Fuxman, Elham Fazli, and Renee J. Miller. “ConQuer: Efficient Management of Inconsistent Databases”. In *SIGMOD*, pages 155–166, 2005.
- [13] H. Galhardas, D. Florescu, D. Shasha, and E Simon. “AJAX: An Extensible Data Cleaning Tool”. In *SIGMOD*, 2000.
- [14] Gösta Grahne. “*The Problem of Incomplete Information in Relational Databases*”.
- [15] Gösta Grahne. “Dependency Satisfaction in Databases with Incomplete Information”. In *VLDB*, pages 37–45, 1984.
- [16] Mauricio A. Hernandez and Salvatore J. Stolfo. “Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem”. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.
- [17] T. Imielinski and W. Lipski. “Incomplete Information in Relational Databases”. *Journal of ACM*, 31:761–791, 1984.
- [18] T. Imielinski, S. Naqvi, and K. Vadaparty. “Incomplete Objects — a Data Model for Design and Planning Applications”. In *SIGMOD*, pages 288–297, Denver, Colorado, 1991.
- [19] Michael Keller. “*Updating Relational Databases through Views*”. PhD thesis, Stanford, CA, USA, 1985.
- [20] Leonid Libkin. “*Aspects of Partial Information in Databases*”. PhD thesis, Philadelphia, PA, USA, 1995.
- [21] Leonid Libkin and Limsoon Wong. “Semantic Representations and Query Languages for OR-Sets”. In *PODS*, pages 37–48, 1993.
- [22] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. “Testing Implications of Data Dependencies”. *ACM Transactions on Database Systems*, 4(4):455–469, 1979.
- [23] Erhard Rahm and Hong Hai Do. “Erhard Data Cleaning: Problems and Current Approaches”. *IEEE Data Engineering Bulletin*, 2000.

- [24] V. Raman and J.M. Hellerstein. “Potter’s Wheel: An Interactive Data Cleaning System”. In *VLDB*, pages 381–390, 2001.
- [25] Steven Ruggles, Matthew Sobek, Trent Alexander, Catherine A. Fitch, Ronald Goeken, Patricia Kelly Hall, Miriam King, and Chad Ronnander. “Integrated Public Use Microdata Series: Version 3.0”, 2004. <http://www.ipums.org>.
- [26] Jeffrey D. Ullman. “*Principles of Database & Knowledge-Base Systems Vol. 2: The New Technologies*”. Computer Science Press, 1989.
- [27] Moshe Y. Vardi. “The Complexity of Relational Query Languages”. In *14th Annual ACM Symposium on Theory of Computing (STOC’82)*, pages 137–146, San Francisco, CA USA, May 1982.