Malleable* Schemas: A Preliminary Report

Xin Dong
University of Washington
Seattle, WA 98195
lunadong@cs.washington.edu

Alon Halevy
University of Washington
Seattle, WA 98195
alon@cs.washington.edu

ABSTRACT

Large-scale information integration, and in particular, search on the World Wide Web, is pushing the limits on the combination of structured data and unstructured data. By its very nature, as we combine a large number of information sources, our ability to model the domain in a completely structured way diminishes. We argue that in order to build applications that combine structured and unstructured data, there is a need for a new modeling tool. We consider the question of modeling an application domain whose data may be partially structured and partially unstructured. In particular, we are concerned with applications where the *border* between the structured and unstructured parts of the data is not well defined, not well known in advance, or may evolve over time.

We propose the concept of *malleable schemas* as a modeling tool that enables incorporating both structured and unstructured data from the very beginning, and evolving one's model as it becomes more structured. A malleable schema begins the same way as a traditional schema, but at certain points gradually becomes vague, and we use keywords to describe schema elements such as classes and properties. The important aspect of malleable schemas is that a modeler can *capture* the important aspects of the domain at modeling time without having to commit to a very strict schema. The vague parts of the schema can later evolve to have more structure, or can remain as such. Users can pose queries in which references to schema elements can be imprecise, and the query processor will consider closely related schema elements as well.

1. INTRODUCTION

There has been significant interest recently in combining techniques from data management and information retrieval (as surveyed in [4]). The underlying reason is that knowledge workers in enterprises are frequently required to analyze data that exist partially in structured databases and partially in content management systems or other repositories of unstructured data. Similarly, the WWW is a repository of both structured and unstructured sources (webforms and webpages). To support the querying needs in these applications we should be able to seamlessly query both structured and unstructured data, and consider query paradigms that involve both ranking answers and structure based (SQL-like) conditions on query answers.

Previous work in this area focused on dealing with hybrid data *after* the fact. That is, it is assumed that we already have some set of structured data and another set of unstructured data, and the goal is to manage it and query seamlessly.

This paper looks at the entire process of building an application that involves both structured and unstructured data. We ask the following basic question: how do we model data for an application that will involve both structured and unstructured data? In particular, we are concerned with the case where the *border* between the structured and unstructured parts of the data is not well defined, and may evolve over time.

When we start modeling a domain, we typically want to model it as precisely as possible by defining its structure with a schema (or possibly a more expressive modeling paradigm such as an ontology). However, in the process of modeling we may realize the following. First, we may not be able to give a precise model of the domain, either because we don't know what it is or because one does not exist. Second, we may prefer not to model the domain in such level of detail because an overly complex model may be a burden on the users. Third, there are parts of the domain we may want to leave unstructured for the time being.

To address these needs, we propose the concept of *malleable schemas* as a modeling tool that enables incorporating both structured and unstructured data from the very beginning, and evolving one's model as it becomes more structured. A malleable schema begins the same way as a traditional schema, but at certain points gradually becomes vague. The important aspect of malleable schemas is that a modeler can *capture* the important aspects of the domain at modeling time without having to commit to a very strict schema. The vague parts of the schema can later evolve to have more structure, or can remain as such. Users can pose queries in which references to schema elements can be im-

*`Merriam-Webster:` *Malleable* – 1: capable of being extended or shaped by beating with a hammer or by the pressure of rollers 2a: capable of being altered or controlled by outside forces or influences b: having a capacity for adaptive change

Copyright is held by the author/owner.
Eighth International Workshop on the Web and Databases (WebDB 2005), June 16-17, 2005, Baltimore, Maryland.

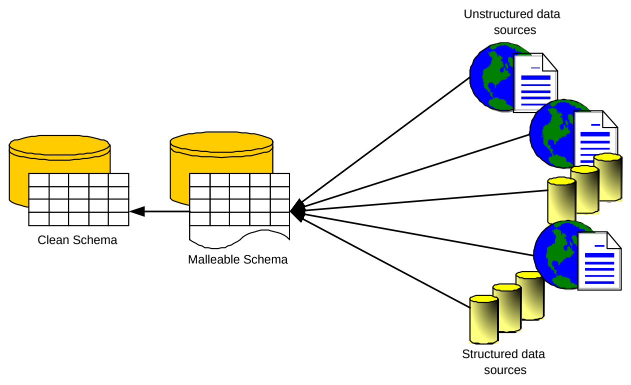

Figure 1: When someone is trying to create a schema for a domain to integrate both structured and unstructured data from a variety of data sources, malleable schemas can help her *capture* the important aspects of the domain at modeling time without having to commit to a very strict schema. The vague parts of the schema can later evolve to have more structure, or can remain as such.

precise, and the query processor will consider closely related schema elements as well. Figure 1 depicts the key idea of malleable schemas.

The concept of malleable schemas evolved from several different applications we have been considering recently: (1) personal information management (PIM) [8], where we constantly model both structured and unstructured data and the model of the domain needs to be very easy to use, (2) information integration on the web [3, 9, 18], where the diversity of information sources does not allow creating a single mediated schema to which everything cleanly maps, and (3) biomedical informatics [21], where our understanding of the domain is constantly evolving from multiple different views and sources.

Large-scale information integration remains one of the important challenges in web data management. By its very nature, as we combine a large number of information sources, our ability to model the domain in a completely structured way diminishes. We argue that the marriage of structured and unstructured data is crucial for building robust integration systems, and the modeling questions that arise are key to the success of such systems. This paper presents our initial work on malleable schemas. We motivate the concept with examples, present an initial formal model, and discuss the implementation challenges.

Related work

There has been a significant body of work on supporting keyword search in databases [15, 1, 16], result ranking [2, 13, 12], and approximate queries [20, 23, 10, 11, 5]. They all assume that the model of the data is precise, but we want to add flexibility in the queries. In contrast, our goal is to allow the model itself to be imprecise in certain ways. Probabilistic databases [24, 7] (and formalisms such as Bayesian Networks) allow imprecision about facts in the database, but the model of the domain is still a precise one.

The work closest to ours is the XXL Query Engine [22], where the queries allow for imprecise references to schema elements. The idea there is that the user will query a large *collection* of XML DTDs, and there is no unifying DTD for all of them. Malleable schemas, in contrast, offer a middle point between a collection of schemas/DTDs (or a corpus [19]) in a domain and a single crisp schema for that domain. The idea of a malleable schema is that someone *is* trying to create a schema for the domain, but in the process of doing so needs to introduce (possibly temporarily) some imprecision into the model. We expect to leverage some of the techniques in [22, 23] in our query processing engine.

Outline: We first present motivating examples for malleable schemas. Section 3 defines a bare-bones formalism that includes malleable classes and properties, and describes the semantics of querying. Section 4 describes several extensions to the basic model, and Section 5 discusses implementation issues. Section 6 concludes.

2. MOTIVATING EXAMPLES

We present two motivating examples for malleable schemas taken from our application domains: information integration on the web and personal information management. The intuition underlying malleable schemas is the following. A traditional schema is a very structured specification of the domain of interest. It assumes that you *know* the structure that you're trying to capture and that it *can* be specified. Malleable schemas are meant for contexts in which one or more of the following hold:

- There is no obvious structure for the domain, and therefore our model of the domain needs to be vague at certain places.
- The structure of the domain is not completely known at modeling time, and may become clearer as the application evolves and the user needs clarify.
- The structure is inherently evolving over time because the domain is extremely complicated and itself the subject of study (e.g, biomedical informatics). Consequently, by nature there will always be parts of the domain that are not precisely modeled.
- A complete structure of the domain would be too complicated for a user to interact with. For example, trying to model every detail of items found on one's desktop in a PIM system would be too overwhelming for a typical user, and maintaining the model would also be impractical.
- The borders between the structured and unstructured parts of the data are fuzzy, and therefore the modeling paradigm needs to support smoother transitions between the two parts.

The idea of malleable schemas is the following. A modeler starts out creating a schema of a domain intending to capture the domain as precisely as possible. However, at certain points in the modeling process, the schema can become less precise. Malleable schemas provide a mechanism by which the modeler can *capture* the imprecise aspects of the domain during the modeling phase in a well principled fashion. Malleable schemas allow the modeler to capture these aspects using keywords, but tell the system that these keywords are meant to capture elements of the schema, rather than being arbitrary keyword fields.

In the discussion below we assume a very simple data model: our domain is comprised of objects (with ID's). Objects have properties – we distinguish between relationships that relate pairs of objects and attributes that relate objects with ground values. Objects are members of classes that can form a hierarchy. We assume objects can belong to multiple classes.

Example 1. Consider building an information integration system for web sources, whose goal is to answer queries from multiple databases available on the web (e.g., querying multiple real-estate sites).

You begin modeling the domain by trying to capture the salient aspects of real-estate that appear in the sources, and such that you'll be able to pose meaningful queries on as many sites as possible. As an example, you create the class RealEstate, intended to denote real-estate objects for sale or rental. In an ideal world (which seems likely when you start building the application), there would be some obvious sub-classes of RealEstate (such as houses, condo's) that you would incorporate into the model. However, after inspecting several sites you realize that there are many more sub-classes, and the relationship between them is not clear. Furthermore, different sites organize real-estate objects in varying ways, and the concepts used in one place overlap but don't correspond directly with concepts used elsewhere. For example, you may encounter vacation rentals, short-term rental and sublets. As a consequence, you cannot create a model of real-estate such that there would be a clean mapping between your categories and those used in the sources. In short, there is no single way of identifying all the subclasses of RealEstate.

What you would like to do now is to create a set of subclasses, each described by words or a phrase (typically found as menu items on a real-estate search form on the web). The subclasses will not necessarily be disjoint from each other; in fact, there may be overlaps between the classes. Later on in the life of the application, after having seen many real-estate listings and user queries, you may decide to impose more structure on the subclasses of RealEstate.

In principle, you could do this by creating a property for the class RealEstate called RealEstateType, and have keywords or phrases be the content of that property. However, while doing so could be a way of *implementing* malleable schemas (see Section 5), it has several disadvantages from the modeling perspective because the system does not *know* that these keywords are identifying subclasses of RealEstate. Specifically, (1) you would like to refer to these subclasses in queries in the same way as you refer to other sub-classes, (2) later on you would like to evolve the schema (possibly with the help of the system) to create a more crisp class hierarchy, and (3) you may want to create subclasses for these classes as well. Hence, in a sense, you want to create a new keyword property, but you want the system to know that it is identifying subclasses of an existing class.

This type of example is extremely common in information integration applications that involve many independently developed sources. By nature these domains are complicated and there is no obvious single way to model them. Different categorizations arise because site builders have different views of the world, and often because of natural geographical differences. In addition, large-scale information integration fundamentally pushes on the limits being able to model a domain with a single structured representation. □

Example 2. The following example illustrates that the same idea can be applied to properties in the schema. Consider the domain of personal information management, where the goal is to offer users a logical view of the information on their desktops [8]. (Note that in practice this logical view is created *automatically* without any investment by the user).

Suppose you are creating a schema for information that people store on their desktops. You create a class called Project, and a property called Participant. But soon you realize that not all participants are equal, and you have various kinds of participation modes. For example, you may have a programmer on the project, a member in the initial planning phases, advice-giver, etc. You cannot anticipate all the possible participation modes nor classify them very crisply. Hence, you would like to create sub-properties of Participant so you can at least *capture* some of the information about the types of participation, and have these sub-properties described by keywords. Note that in this example, even if you could create a clear description of all the types of participation, you may not want to do so because the model will be too complex for users to understand.

This example is not possible to implement with yet another keyword attribute as we did in Example 1. Suppose you create a text property called ParticipationType. The question is then what object to attach it to. It does not suffice to attach it to the participant object because it is not a property of that object, but of the relationship of that object to the project. In principle, the keyword is expressing a relationship between two objects in the domain, and the only way to do that in the object-oriented model we are considering is with a property. Of course, even if you could express ParticipationType somehow, all the disadvantages mentioned in Example 1 still hold. □

Note that one of the early purported advantages of XML is that you can add tags (corresponding to properties and classes) at will. Even ignoring for a moment that XML has evolved to be mostly guided by schemas, XML is, again, a possible implementation avenue for malleable schemas. However, our focus is on the modeling aspects – trying to create a schema for a domain while capturing the vague aspects and evolving the schema with time.

3. FORMALIZING MALLEABLE SCHEMAS

We now describe a formal model for malleable schemas. We focus on the main constructs, and then mention several extensions in Section 4.

3.1 The data model

We frame our discussion in the context of a very simple schema formalism, close in spirit to object-oriented schemas. There have been a plethora of object-oriented modeling languages suggested in the literature. Our goal is not to argue for one or the other. Instead, we chose a set of features from these languages that are important for our discussion, and our focus is on adding malleable features to the formalism.

We model the domain using objects and properties. Each property has a domain and a range, where the domain is a set of classes, and the range is either a set of classes or a set of ground values. We distinguish between two types of properties: relationships, whose ranges are sets of classes, and attributes, whose ranges are sets of ground values. In other words, a relationship is a binary relation between a pair of objects, while an attribute is a binary relation between an object and a ground value. We denote classes by

$C_1, \ldots, C_m$, and properties by $P_1, \ldots, P_n$. In what follows, we refer to classes and properties collectively as *elements.*

We support class hierarchies and property hierarchies, which model the IS-A relationships. For example, Condo is a sub-class of RealEstate, and programmer is a sub-property of participant. Specifically, $C_i \sqsubseteq C_j$ denotes that C_i is a sub-class of C_j, and $P_i \sqsubseteq P_j$ denotes that P_i is a sub-property of P_j. We assume that the classes form a directed acyclic graph, as do the properties. Note that a sub-class inherits properties from its parent classes. That is, if C_1 is in the domain (resp. range) of P_1, and $C_2 \sqsubseteq C_1$, then C_2 is also in the domain (resp. range) of P_1. The domain and range of a sub-property can be sub-classes of those of its parent-properties. Specifically, if C_1 and C_2 are in the domain (resp. range) of P_1 and P_2 respectively, and $P_1 \sqsubseteq P_2$, then $C_1 \sqsubseteq C_2$.

The malleable schema elements: The malleable elements look exactly the same as the other schema elements, except for the following (mostly conceptual) differences:

- While the name of a regular class or property is typically a carefully chosen string, the names of schema elements can be keywords or phrases, and those are often obtained from external sources. Later we will extend malleable schema elements to include also regular expressions (Section 4).
- For simplicity, we restrict malleable elements to appear only on the left-hand side of $\sqsubseteq$ inclusions. We can easily extend and allow malleable elements on the right-hand side (i.e., have a sub-class element for a malleable element).
- They are marked as malleable. (This is not a requirement, but it may be important for future schema evolution.)

We refer to malleable elements as either *malleable classes* or *malleable properties.* Note that the same name can be both a malleable property and a malleable class, though they are treated as two distinct elements in the schema.

While from a formal point of view malleable schema elements are not so different from ordinary ones, the important point to emphasize is how they are used in the modeling process. The typical process of modeling a domain assumes that we are trying to come up with a very clean model, and hence choose our schema names carefully. In contrast, the malleable schema elements are meant for the cases where we cannot (maybe temporarily) model the domain cleanly, and so we capture certain aspects using keywords. Hence, by nature we may have many overlapping malleable classes or properties (possibly even identical ones called differently), and there will typically be relatively many malleable sub-classes for a class (or sub-properties of a property).

Example 3. Continuing with example 1, suppose we define the following malleable sub-classes of RealEstate: VacationRental, ShortTermRental, and Sublet. In addition, we define the following malleable sub-properties of contactPerson: agent, leaseAgent, and rentalClerk. Note that it is hard to precisely define the relationships between these malleable schema elements (for example, VocationRental, ShortTermRental and Sublet can largely overlap), but we would like to capture them in the model. Formally, we have:

- VacationRental, ShortTermRental, Sublet $\sqsubseteq$ RealEstate
- agent, leaseAgent, rentalClerk $\sqsubseteq$ contactPerson

3.2 Queries

In our discussion of queries we do not pin down a specific query language. Instead, we describe the principles of incorporating malleable schemas into a given query language. Our goal is to modify a given query language as minimally as possible.

There are two changes we make to the query language. First, wherever we can refer to a class (resp. property) we allow the query to refer to a malleable class (resp. malleable property). Second, we distinguish between *precise* references in the query and *imprecise* ones. We denote imprecise references by $\sim K$, where K is either a class or a property.

Example 4. Consider the following query that asks for short term rentals in the Tahoe area. Note that we have an imprecise reference to ShortTermRental and to leaseAgent.

Q : SELECT city, price, $\sim$ leaseAgent
FROM $\sim$ ShortTermRental
WHERE location="Tahoe"

□

A query that only makes precise references is answered in exactly the same way as it would be otherwise. That is, we treat every malleable schema element as a normal schema element.

The interesting case is when the query can make imprecise references to the malleable schema elements. Intuitively, when we have a reference $\sim K$, we want to refer to all elements in the schema that are *similar* to the element K. We do not make the definition of similarity part of the query language, since it depends on the particular context of the application. For example, the following types of similarities can be employed:

- **Term similarity:** Schema names can be compared by using some string distance such as the Levenstein measure [6], or according to some lexical references, such as Wordnet [25], or by the term usage similarity computed with some TF/IDF measure on a corpus of documents on the application domain, or using the combination of any of the above.
- **Instance similarity:** Similarity can be estimated by gleaning information from the instances in the database. For example, if the instances of Apartment and Flat tend to have very similar characteristics, we may deem them to be similar.
- **Structural similarity:** Here, the similarity of two elements can be determined by their context. We can compare the super-elements, sub-elements, and sibling-elements of two elements. For example, if two elements have very similar sub-elements, chances are higher that they are similar. Also, two sibling elements can be similar as they might overlap.
- **Schema-corpus similarity:** There have been several pieces of recent work exploring the use of schema corpora for tasks such as schema matching and mediated schema creation [14, 19]. The underlying idea in these works is to leverage statistics on large collections of schemas in order to determine similarity between attributes from disparate schemas. The same idea can be applied here, where instead of similarity

between disparate schemas we consider similarity between terms in the same malleable schema. In fact, a malleable schema can be viewed as an intermediate point in the evolution of a corpus of schemas into a traditional schema.

Note that in principle, the names of schema elements appearing in imprecise references do not even have to be in the schema. Hence, malleable schemas are attractive in cases where users are querying unfamiliar (or very complex) schemas.

Reformulating queries over malleable schemas: Given a similarity measure over malleable schema elements, the next issue is how to *expand* a query over a malleable schema to get the intended answer.

In the simplest case, query reformulation amounts to expanding to a union query. For example, if in Q the reference to leaseAgent were a precise one, then we simply need to create a union query that considers both ShortTermRental and VacationRental, assuming they were deemed to be similar sub-classes. However, this is not the end of the story. First, it may be the case that VacationRental does not have a leaseAgent property. In that case we need to pose the query so that the tuples coming from VacationRental do not have the column for leaseAgent (otherwise the query will be invalid). Second, since Q does have an imprecise reference to leaseAgent, we need to check several combinations, resulting in the following query:

```
Q' : SELECT city, price, leaseAgent
     FROM ShortTermRental
     WHERE location="Tahoe"
     OR
     SELECT city, price, rentalClerk
     FROM ShortTermRental
     WHERE location="Tahoe"
     OR
     SELECT city, price, leaseAgent
     FROM VacationRental
     WHERE location="Tahoe"
     OR
     SELECT city, price, rentalClerk
     FROM VacationRental
     WHERE location="Tahoe"
```

Some of these subqueries may not be valid, and therefore need to be pruned. Furthermore, some of the subqueries can be combined (returning four attributes in each query block).

Finally, we note that there has been significant work on trying to rank answers of queries posed over combinations of structured and unstructured data [4]. We do not go into that issue here, and believe that it is largely orthogonal to the concept of malleable schemas.

Querying the schema: In addition to allowing queries on the instances, we allow queries on the schema (e.g., in the spirit of [17]), as the user might want to know the relationship between the schema elements to help evolve the schema. Given class C, the user can ask for C's parent-classes and sub-classes, and more importantly, for classes that are similar to C. The same queries can be posed for properties too.

4. EXTENSIONS

We now briefly mention several extensions to our basic model for malleable schemas.

Malleable property chains: This extension is a powerful generalization of malleable properties. In addition to the imprecision that can be captured with malleable properties, malleable chains can capture varying *structures* of data. For example, when we integrate information about people from multiple sources, not only do we have different properties for people, but they may be structured differently (e.g., the nesting structure of name, address, etc.). Note that in querying, the similarity among chains compares not only each of the properties in the chain, but also global aspects of the chain. Hence, for example, we may consider two chains with *different* lengths to be similar (e.g., phoneNo with contact/phone), or we may consider the concatenation of two chains to be similar to another chain(e.g., name/firstName and name/lastName with fullName).

Element names as regular expressions:[1] Often there is more structure to the set of sub-classes (or sub-properties) we want to define, and this structure can be described by regular expressions. For example, we may want to create properties *Agent to denote any kind of agent, and define *Agent $\sqsubseteq$ agent to specify that they are sub-properties of agent. In this way we may help *identify* various properties that we want to be agent-related properties.

Malleable values: We often capture aspects of objects in our model with values. For example, when modeling web sites for an integration application, we may have an attribute topic that is assigned one of several values (e.g., BusinessRelated, KidsRelated, Shopping). Formally, these can also be specified as sub-classes in the model, but it is sometimes easier to model such distinctions with values. Hence, we can also support *malleable values.* For example, suppose you created a value BusinessRelated for modeling web sites that have content related to business. However, you then realize that you are not quite sure what you precisely mean by this category. There are web sites that offer articles about business, reviews of business and products, and sites about business people. You can create a description attribute that can have these values and maybe later evolve them into categorization as well.

5. IMPLEMENTATION

We are currently implementing a prototype modeling and querying tool for malleable schemas. We are implementing it over a relational database, though most of the principles of the implementation should carry over to XML, object-relational systems or data integration systems. The details of the implementation are beyond the scope of this paper. We briefly describe its main components below.

- *Modeling*: The modeling tool enables the modeler to create a malleable schema (in terms of classes and properties). The tool also allows to query the model itself in the process of modeling. For example, when the modeler creates a sub-class, she may want to query for similar sub-classes that are already in the schema.

[1]We thank Gerhard Weikum for this idea.

- *Translation to relational schema*: We take the malleable schema and create a malleable relational schema for storing the data.
- *Query reformulation*: Given a query over the relational schema, we translate it into a set of SQL queries that can be posed over the database. The translation process obtains similarity measures between schema elements from an external module.
- *Ranking*: The ranking of the answers in the result considers two factors. First, the set of queries generated by the query reformulator is ordered by the similarity of the schema elements. Second, when we actually see the tuples in the result, we may further refine the ordering of the answers.

6. CONCLUSIONS AND FUTURE WORK

We described malleable schemas, a conceptual tool for modeling in applications that involve both structured and unstructured data. The key idea underlying malleable schemas is that the modeler should be able to capture all the aspects of the domain without having to commit to a clean schema immediately. We argue that such a capability is crucial in applications that combine data from a large number of sources since it is typically impossible to create a clean single schema from the start. In fact, malleable schemas can be viewed as an intermediate point in the evolution of a large collection of schemas into a single coherent schema for a domain. Malleable schemas raise several interesting semantic issues, as well as challenges for efficient query processing and automatically evolving a malleable schema to more structured schema.

Acknowledgments

We thank the anonymous reviewers for their insightful comments. The research was funded by NSF CAREER Grant 9985114, NSF ITR Grant 0205635, and DARPA Contract 03-000225.

7. REFERENCES

[1] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *ICDE*, 2002.

[2] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. In *Proc. of CIDR*, 2003.

[3] K. C.-C. Chang, B. He, and Z. Zhang. Toward large scale integration: Building a metaquerier over databases on the web. In *CIDR*, 2005.

[4] S. Chaudhuri, R. Ramakrishnan, and G. Weikum. Integrating DB and IR technologies: What is the sound of one hand clapping? In *Proc. of CIDR*, 2005.

[5] W. W. Cohen. Data integration using similairty joins and a word-based information representation language. *ACM Transactions on Information Systems*, 18(3):288–321, 2000.

[6] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWEB*, pages 73–78, 2003.

[7] N. Dalvi and D. Suciu. Answering queries from statistics and probabilistic views. In *VLDB*, 2005.

[8] X. Dong and A. Halevy. A Platform for Personal Information Management and Integration. In *Proc. of CIDR*, 2005.

[9] X. Dong, J. Madhavan, and A. Halevy. Mining structures for semantics. *ACM SIGKDD Explorations Newsletter*, 6:53–60, 2004.

[10] R. Fagin. Fuzzy queries in multimedia database systems. In *PODS*, 1998.

[11] L. Gravano, P. G. Ipeirotis, H.V.Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, 2001.

[12] L. Guo, J. Shanmugasundaram, K. Beyer, and E. Shekita. Structured value ranking in update-intensive relational databases. In *ICDE*, 2005.

[13] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *SIGMOD*, 2003.

[14] B. He and K. C.-C. Chang. Statistical schema matching across web query interfaces. In *Proc. of SIGMOD*, 2003.

[15] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style keyword search over relational databases. In *VLDB*, 2003.

[16] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *VLDB*, 2002.

[17] L. V. S. Lakshmanan, F. Sadri, and S. N. Subramanian. Schemasql: An extension to sql for multidatabase interoperability. *ACM Transactions on Database Systems*, 26(4):476–519, 2001.

[18] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB*, pages 251–262, Bombay, India, 1996.

[19] J. Madhavan, P. Bernstein, A. Doan, and A. Halevy. Corpus-basd schema matching. In *ICDE*, 2005.

[20] A. Marian, S. Amer-Yahia, N. Koudas, and D. Srivastava. Adaptive query processing of top-k queries in XML. In *ICDE*, 2005.

[21] P. Mork, A. Halevy, and P. Tarczy-Hornoch. A model for data integration systems of biomedical data applied to online genetic databases. In *AMIA*, 2001.

[22] A. Theobald and G. Weikum. Adding relevance to XML. *Lecture Notes in Computer Science*, 1997:105–124, 2000.

[23] A. Theobald and G. Weikum. The index-based XXL search engine for querying XML data with relevance ranking. In *EDT*, 2002.

[24] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, 2005.

[25] Wordnet. http://www.cogsci.princeton.edu/ wn/.