Provenance in ORCHESTRA

Todd J. Green*
University of California, Davis
Davis, CA USA
green@cs.ucdavis.edu

Grigoris Karvounarakis*
LogicBlox
Atlanta, GA
ICS-FORTH
Heraklion, Greece
gregkar@gmail.com

Zachary G. Ives Val Tannen
University of Pennsylvania
Philadelphia, PA USA
{zives,val}@cis.upenn.edu

Abstract

Sharing structured data today requires agreeing on a standard schema, then mapping and cleaning all of the data to achieve a single queriable mediated instance. However, for settings in which structured data is collaboratively authored by a large community, such as in the sciences, there is seldom consensus about how the data should be represented, what is correct, and which sources are authoritative. Moreover, such data is dynamic: it is frequently updated, cleaned, and annotated. The ORCHESTRA *collaborative data sharing system develops a new architecture and consistency model for such settings, based on the needs of data sharing in the life sciences. A key aspect of* ORCHESTRA*'s design is that the provenance of data is recorded at every step. In this paper we describe* ORCHESTRA*'s provenance model and architecture, emphasizing its integral use of provenance in enforcing trust policies and translating updates efficiently.*

1 Introduction

One of the most elusive goals of the data integration field has been supporting sharing across large, heterogeneous populations. While data integration and its variants (e.g., data exchange [9] and warehousing) are being adopted in the enterprise, little progress has been made in integrating broader communities. Yet the need for sharing data across large communities is increasing: most of the physical and life sciences have become data-driven as they have attempted to tackle larger questions. The field of bioinformatics, for instance, has a plethora of different databases, each providing a different perspective on a collection of organisms, genes, proteins, diseases, and so on. Associations exist between the different databases' data (e.g., links between genes and proteins, or gene homologs between species). Unfortunately, data in this domain is surprisingly difficult to integrate, primarily because conventional data integration techniques require the development of a single global schema and complete global data consistency. Designing one schema for an entire community like systems biology is arduous, involves many revisions, and requires a central administrator.

Even more problematic is the fact that the data or associations in different databases are often contradictory, forcing individual biologists to choose values from databases they personally consider most authoritative or trusted [23]. Such inconsistencies are not handled by data integration tools, since there is no consensus or "clean" version of the data. Thus, scientists simply make their databases publicly downloadable, so users can copy and convert them into a local format (using ad hoc scripts). Meanwhile the original data sources continue to be edited. In some cases the data providers publish weekly or monthly lists of updates (*deltas*) to help others keep synchronized. Today, few participants, except those with direct replicas, can actually exploit such deltas;

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Work performed while at the University of Pennsylvania

hence, once data has been copied to a dissimilar database, it begins to diverge from the original.

In order to provide collaborating scientists, organizations, and end users with the tools they need to share and revise structured data, our group has developed a new architecture we term *collaborative data sharing systems* [18] (CDSSs) and the first implementation of a CDSS in the form of the ORCHESTRA system. The CDSS provides a principled semantics for exchanging data and updates among autonomous sites, which extends the data integration approach to encompass scientific data sharing practices and requirements—in a way that also generalizes to many other settings. The CDSS models the exchange of data among sites as *update exchange* among peers, which is subject to transformation (via schema mappings), filtering (based on trust policies regarding source authority), and local revision or replacement of data. As a prerequisite to assessing trust, the CDSS records the derivation of all exchanged data, i.e., its *provenance* or lineage [5].

In this paper, we provide an overview of the basic operation of the ORCHESTRA CDSS, emphasizing its use of data provenance for enforcing trust policies and for performing update exchange incrementally and efficiently. This paper summarizes our main results from [14, 20, 21]; we mention further aspects of the system in Section 6.

2 Overview of CDSS and ORCHESTRA

The CDSS model represents a natural next step in the evolution of ideas from data integration, peer data management systems [17] (PDMS), and data exchange [9]. The PDMS model removes data integration's requirement of a single central schema: rather, the PDMS supports greater flexibility and schema evolution through *multiple* mediated schemas (peers) interrelated by compositional schema mappings. Along a different dimension, data exchange alleviates the tight coupling between data sources and queriable target data instance: it enables *materialization* of data at the target, such that queries over the target will be given the same answers as in traditional virtual data integration.

As in the PDMS, the CDSS allows each data sharing participant to have an independent schema, related to other schemas via compositional schema mappings. However, in a CDSS each participant (peer) materializes its own database instance (as in data exchange). Through a variety of novel techniques we sketch in this paper and describe in detail in [14, 20, 21], the CDSS enables the peer to *autonomously control* what data is in the instance by applying updates and policies about which data is most trusted.[1]

2.1 The topology of data sharing: specifying how peers are related

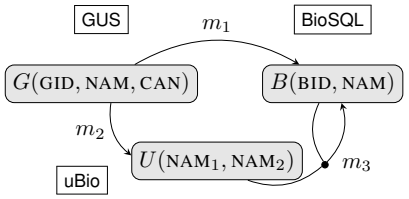
A CDSS collaboration begins with a set of *peer* databases, each with its own schema and local data instance. Each peer's users pose queries and make updates directly to its local data instance.

Example 1: Consider (see Figure 1) a bioinformatics collaboration scenario based on databases of interest to affiliates of the Penn Center for Bioinformatics. GUS, the Genomics Unified Schema covers gene expression, protein, and taxon (organism) information; BioSQL, affiliated with the BioPerl project, covers very similar concepts; and a third schema, uBio, establishes synonyms among taxa.[2] Instances of these databases contain taxon information that is autonomously maintained but of mutual interest to the others. For the purposes of our example we show only one relational table in each of the schemas, as follows. Peer GUS associates taxon identifiers, scientific names, and what it considers *canonical* scientific names via relation $G(\text{GID}, \text{NAM}, \text{CAN})$; peer BioSQL associates its own taxon identifiers with scientific names via relation $B(\text{BID}, \text{NAM})$; and peer uBio records synonyms of scientific names via relation $U(\text{NAM}_1, \text{NAM}_2)$. □

The participants of the CDSS collaboration specify relationships among their databases using *schema mappings*.

[1] Our implementation assumes relational schemas, but the CDSS model extends naturally to other data models such as XML [10].

[2] The real databases can be found at `http://www.gusdb.org`, `http://bioperl.org`, and `http://www.ubio.org`.



$$(m_1) \quad G(g,n,c) \rightarrow \exists b\, B(b,n)$$
$$(m_2) \quad G(g,n,c) \rightarrow U(n,c)$$
$$(m_3) \quad B(b,n_1) \wedge U(n_1,n_2) \rightarrow B(b,n_2)$$

Figure 1: Mappings among three bioinformatics databases

G :

GID	NAM	CAN
828917	*Oscinella frit*	*Drosophila melanogaster*
2616529	*Musca domestica*	*Musca domestica*

U :

NAM_1	NAM_2

B :

BID	NAM
4472	*Periplaneta americana*

(a) Tables from GUS, uBio, and BioSQL before update exchange. (U is empty.)

U :

NAM_1	NAM_2
Oscinella frit	*Drosophila melanogaster*
Musca domestica	*Musca domestica*

B :

BID	NAM
4472	*Periplaneta americana*
$\perp_1$	*Oscinella frit*
$\perp_1$	*Drosophila melanogaster*
$\perp_2$	*Musca domestica*

(b) Tables after update exchange. Shaded rows indicate newly-inserted tuples. (G is unchanged.)

Figure 2: Bioinformatics database instances before and after update exchange

Example 2: Continuing with Example 1, suppose it is agreed in this collaboration that certain data in GUS should also be in BioSQL. This represented in Figure 1 by the arc labeled m_1. The specification $G(g,n,c) \rightarrow \exists b\, B(b,n)$ associated with m_1 is read as follows: if (g,n,c) is in table G, the value n must also be in some tuple (b,n) of table B, although the value b in such a tuple is not determined. The specification just says that there must be such a b and this is represented by the existential quantification $\exists b$. Here m_1 is an example of *schema mapping*. Two other mappings appear in Figure 1. Peer uBio should also have some of GUS's data, as specified by m_2. Mapping m_3 is quite interesting: it stipulates data in BioSQL based on data in uBio but also on data *already* in BioSQL. As seen in m_3, relations from multiple peers may occur on either side. We also see that individual mappings can be "recursive" and that cycles are allowed in the graph of mappings. □

Schema mappings (expressed in ORCHESTRA using the well-known formalism of *tuple-generating dependencies* or tgds [2]) are logical assertions that the data instances at various peers are expected to jointly *satisfy*. As in data exchange [9], a large class of mapping graph cycles can be handled safely, while certain complex examples cause problems and are prohibited.

In summary, every CDSS specification begins with a collection of peers/participants, each with its own relational schema, and a collection of schema mappings among peers. Like the schemas, the mappings are designed by the administrators of the peers to which data is to be imported. (In addition, administrators also supply *trust policies*, described in Section 3.) By joining ORCHESTRA, the participants agree to share the data from their local databases. Mappings and trust policies provide a more detailed declarative specification of how data is to be shared.

2.2 The semantics of query answers, and what must be materialized at each peer

Given a CDSS specification of peers and schema mappings, the question arises of how data should be propagated using the declarative mappings, and what should be materialized at the target peer. Clearly, a user query posed over a peer's materialized instance should provide answers using relevant data from *all* the peers. This requires materializing at every peer an instance containing not only the peer's locally contributed data, but also additional

facts that *must* be true, given the data at the other peers along with the constraints specified by the mappings. However, while the mappings relating the peers tell us which peer instances are together considered "acceptable," they do not fully specify the complete peer instances to materialize.

In CDSS we follow established practice in data integration, data exchange and incomplete databases [1, 9] and use *certain answers* semantics: a tuple is "certain" if it appears in the query answer no matter what data instances (satisfying the mappings) we apply the query to. In virtual data integration, the certain answers to a query are computed by reformulating the query across all peer instances using the mappings. In CDSS, as in data exchange, we materialize special local instances that can be used to compute the certain answers. This makes query evaluation a fast, local process. We illustrate this with our running example.

Example 3: Suppose the contents of G, U, and B are as shown in Figure 2(a). Note that the mappings of Figure 1 are not satisfied: for example, G contains a tuple $(828917, \text{"}\textit{Oscinella frit}\text{"}, \text{"}\textit{Drosophila melanogaster}\text{"})$ but B does not contain any tuple with "*Oscinella frit*" which is a violation of m_1. Let us patch this by adding to B a tuple $(\perp_1, \text{"}\textit{Oscinella frit}\text{"})$ where $\perp_1$ represents the unknown value specified by $\exists b$ in the mapping m_1. We call $\perp_1$ a *labeled null*. Adding just enough patches to eliminate all violations results in the data in Figure 2(b).[3] □

In order to support edits and accommodate disagreement among participants, CDSS incorporates mechanisms for *local curation* of database instances. This allows CDSS users to modify or delete *any* data in their local database, even data that has been imported from elsewhere via schema mappings. In this way, CDSS users retain full control over the contents of their local database. The technical obstacle in supporting such a feature is how to preserve a notion of semantic consistency with respect to the mappings, when such modifications may introduce violations of the mappings.

Example 4: Refer again to Figure 2(b), and suppose that the curator of BioSQL decides that she is not interested in house flies, and wishes to delete tuple $B(\perp_2, \text{"}\textit{Musca domestica}\text{"})$ from her local instance. Since, as we have seen, this tuple was added to satisfy a mapping, the deletion of the tuple leads to a violation of some mapping. □

To allow such local curation, CDSS stores deleted tuples in *rejection tables*, and newly inserted tuples in *local contribution tables*, and converts user-specified mappings into internal mappings that take the contribution and rejection tables explicitly into account. Alternatively, sometimes local curation corrects *mistakes* in the imported data; if we wish for the corrections to be propagated back to the original source tuples, ORCHESTRA also incorporates facilities for *bidirectional mappings* [20].

3 Trust policies and provenance

CDSS participants often need to filter data based how much they trust it (and its sources). In order to allow the specification of such *provenance-based trust policies*, ORCHESTRA records the *provenance* of exchanged data, i.e., "how data was propagated" during the exchange process.

The most intuitive way to picture CDSS provenance is as a graph having two kinds of nodes: *tuple nodes*, one for each tuple in the system, and *mapping nodes*, where several such nodes can be labeled by the same mapping name. Edges in the graph represent *derivations* of tuples from other tuples using mappings. Special mapping nodes labeled "+" are used to identify original source tuples.

Example 5: Refer to Figure 3, which depicts the provenance graph corresponding to the bioinformatics example from Figure 2 (we have abbreviated the data values to save space). Observe there is a "+" mapping node pointing to $G(26, \textit{Musc.}, \textit{Musc.})$; this indicates that it is one of the source tuples from Figure 2(a), present before update exchange was performed. Next, observe that $G(26, \textit{Musc.}, \textit{Musc.})$ is connected to $U(\textit{Musc.}, \textit{Musc.})$ via a

[3] Note that sometimes patching one violation may introduce a new violation, which in turn must be patched; hence this process is generally an iterative one, however under certain constraints it always terminates.

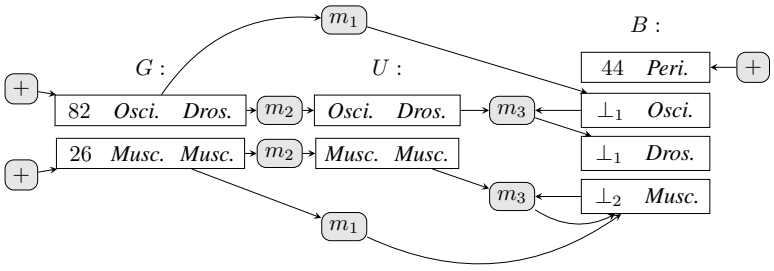


Figure 3: Provenance graph for bioinformatics example

```
EVALUATE TRUST OF {
    FOR [B $x] <$p [] <-+ [$y]
    WHERE $p = m1 OR $p = m3
    INCLUDE PATH [$x] <-+ [$y]
    RETURN $x
} ASSIGNING EACH leaf_node $y {
    CASE $y in G and
        $y.n = "Musc." : SET false
    DEFAULT : SET true
} ASSIGNING EACH mapping $p($z) {
    CASE $p = m3 : SET false
    DEFAULT : set $z
}
```

Figure 4: A provenance query in ProQL

mapping node labeled m_2. This indicates that $U(\textit{Musc.}, \textit{Musc.})$ was derived using $G(26, \textit{Musc.}, \textit{Musc.})$ with m_2. Also, notice that $B(\perp_2, \textit{Musc.})$ has a derivation from $G(26, \textit{Musc.}, \textit{Musc.})$ via mapping m_1. Finally, note that $B(\perp_2, \textit{Musc.})$ also has a second derivation, from itself and $U(\textit{Musc.}, \textit{Musc.})$ via mapping m_3. The graph is thus cyclic, with tuples involved in their own derivations. In general, when mappings are recursive, the provenance graph may have cycles. □

Provenance graphs form the basis of our ORCHESTRA implementation, and [14] shows how they can be computed during data exchange and stored together with exchanged data. Underlying the graphical model, however, is another, equivalent perspective on CDSS provenance, based on a framework of *semiring-annotated relations* [15, 10]. These are algebraic structures that arise naturally with database provenance. Their two operations correspond to joint use of data (represented in the graph model by the incoming edges into a mapping node) and to alternative use of data (represented in the graph model by the incoming edges into a tuple node). *$\mathcal{M}$-semirings* [19] extend those structures with unary functions to capture mappings. The semiring framework has the virtue of uniformly capturing as special cases many other data provenance models from the literature (e.g., *lineage* [7], *why-provenance* [4], and *Trio lineage* [3]). By putting all these models on equal footing, we are able to make precise comparisons of various kinds: e.g., their relative *informativeness*, or their interactions with *query optimization* [12]. In particular, we can explain precisely why ORCHESTRA's provenance is strictly more informative than the other models. Moreover, we see that ORCHESTRA provenance captures the most general semiring computations and thus can be specialized to compute events (hence probabilities), scores, counts, security access levels, etc, as we discuss in Section 5 and [21]. Provenance annotations also assist in formulating efficient algorithms for update exchange, as we explain in Section 4 and [14, 20].

Using such data provenance information, we can compute trust scores for exchanged data, based on CDSS users' beliefs about the trustworthiness of source data and mappings. Trust policies can then be expressed to control the flow of data through mappings, allowing CDSS administrators to specify which tuples are "trusted" or "distrusted," depending on their provenance.

Example 6: Some possible trust policies in our bioinformatics example:

- Peer BioSQL distrusts any tuple $B(b, n)$ if the data came from GUS and $n =$ "*Musca domestica,*" and trusts any tuple from uBio.
- Peer BioSQL distrusts any tuple $B(b, n)$ that came from mapping m_3. □

Once specified, the trust policies are incorporated into the update exchange process: when the updates are being translated into a peer P's schema they are accepted or rejected based on P's trust conditions.

The example above illustrates *Boolean trust policies*, which classify all tuples as either (completely) trusted or (completely) untrusted, depending on their provenance and contents. In fact, CDSS also allows richer forms of *ranked trust policies* in which *trust scores* are computed for tuples indicating various "degrees of trust." When a conflict is detected among data from multiple sources (for example, by a primary key violation), these scores can be used to resolve the conflict by selecting the tuple with the highest trust score and discarding those with

which it conflicts. Fortunately, our same semiring-based provenance information can be used to evaluate either Boolean or ranked trust, as well as—possibly contradictory—trust policies of different CDSS peers without recomputing data exchange solutions [21].

4 Dynamic operation and update exchange

Given that ORCHESTRA performs query answering on locally materialized data instances, this raises the question of data freshness in the peer instances. The CDSS approach sees data sharing as a fundamentally *dynamic* process, with frequent "refreshing" *data updates* that need to be propagated efficiently. This process of dynamic update propagation is called *update exchange*. It is closely related to the classical *view maintenance problem* [16].

Operationally, CDSS functions in a manner reminiscent of *revision control systems*, but with peer-centric conflict resolution strategies. The users located at a peer P query and update the local instance in an "offline" fashion. Their updates are recorded in a *local edit log*. Periodically, upon the initiative of P's administrator, P requests that the CDSS perform an update exchange operation. This *publishes* P's local edit log, making it globally available via central or distributed storage [27]. This also subjects P to the effects of the updates that the other peers have published (since the last time P participated in an update exchange). To determine these effects, the CDSS performs incremental *update translation* using the schema mappings to compute corresponding updates over P's local instance: the translation finds matches of incoming tuples to the mappings' sources, and applies these matchings to the mappings' targets to produce outgoing tuples (recall the "patching" process in Example 3).

Doing this on updates means performing data exchange incrementally, with the goal of maintaining peer instances satisfying the mappings. Therefore, in CDSS the mappings are more than just static specifications: they enable the dynamic process of *propagating* updates.

Example 7: Refer again to Figure 2(b), and suppose now that the curator of uBio updates her database by adding another synonym for the fruit fly: $U($"*Oscinella frit*", "*Oscinella frit Linnaeus*"$)$. When this update is published, it introduces a violation of mapping m_3 that must again be "patched." The update translation process therefore inserts a corresponding tuple into BioSQL: $B(\perp_1,$ "*Oscinella frit Linnaeus*"$)$. □

Propagating the effects of insertions to other peers is a straightforward matter of applying techniques from [16]. Deletions are more complex, both in terms of propagating them *downstream* (to instances mapped from the initial deletion) and optionally *upstream* (to sources of a derived, now-deleted tuple).

Downstream propagation involves identifying which tuples in a derived instance are dependent on the tuple(s) we are deleting and have no alternate derivations from the remaining source tuples. The algorithms of [16] provide one solution, involving deleting and attempting to re-derive all dependent tuples. Provenance enables a more efficient algorithm [14], which determines if a tuple has alternative derivations, avoiding unnecessary deletions and rederivations.

Upstream propagation requires determining the sources of a derived tuple, and removing (some of) them, so that this tuple is no longer derivable through the mappings. This problem is closely related to the *view update problem* [8], and raises the possibility of *side effects*: if a source tuple is removed, additional tuples that were derived from it and were not among the users' deletions may also be deleted. Traditionally, we disallow updates whenever the database integrity constraints do not guarantee side-effect-free behavior (a very restrictive approach in practice). In ORCHESTRA we provide a more flexible approach. We use provenance to check at *run-time* whether some deletion propagation would cause side effects to any other peer specified by the user. We only propagate to the sources those deletions that do not cause side effects, while we handle the remaining deletions through the rejection tables (Section 2.2). This allows much greater levels of update propagation in practice.

5 Querying provenance

To this point, we have described the *internal* uses of provenance in a CDSS. However, provenance is also useful to CDSS end-users as they perform data curation and other manual tasks. Unfortunately, the complexity of provenance graphs can often make it difficult for such users to explore the provenance of data items they are interested in. Moreover, curators may not be interested in the complete provenance of some items, but only derivations involving certain mappings or peers they consider authoritative.

For these reasons, we have designed and implemented *ProQL* [21], a query language for provenance. ProQL can help curators or other CDSS users explore and navigate through provenance graphs, based on as little information as they have in their disposal, through the use of *path expressions*. It also allows them to focus only on parts of the graph which are of interest to them. ProQL builds upon the fact that our provenance model generalizes a variety of annotation computations, such as trust scores, probabilities, counts, security access levels or derivability. In particular, ProQL allows CDSS users to specify "source" annotations for tuple and mapping nodes in these graphs and uses those to compute annotations for derived tuples of interest.

Example 8: The query shown in Figure 4 illustrates some of these features. First, it matches all derivations of tuples in B (of any length, as indicated by $\leftarrow$+) whose last step involves mappings m_1 and m_3. Then, it specifies that any derivations through m_1 as well as tuples in G with name "*Musc.*" are untrusted, while everything else is trusted (essentially, these are the trust policies from Example 6). Applied on the provenance graph of Figure 3, it determines that the tuples $B(44, \text{"}\textit{Peri.}\text{"})$, $B(\bot_1, \text{"}\textit{Osci.}\text{"})$ are trusted, while the remaining tuples in B are untrusted. □

More details about the syntax of ProQL, as well as indexing techniques to optimize provenance query processing can be found in [21].

6 Related work

In this paper we surveyed briefly the core update exchange and provenance facilities of ORCHESTRA, as originally presented in [14, 20], and the provenance query language of [21]. Other major features of the system include a distributed conflict reconciliation algorithm [27], a distributed storage and query engine [28], a keyword-based query system incorporating rankings and user feedback [25]. Related work on incremental update optimization appears in [13]. Data sharing with conflict update resolution policies is studied in [22]. The semiring framework for ORCHESTRA's provenance model was introduced in [15], and further elaborated in [10] and [12].

With hindsight, we find misleading the term "how-provenance" (originating in a footnote of [15] and popularized in [5]). It allows for an interpretation according to which this is yet another provenance model, parallel to why- and where-provenance [4]. As we saw, the semiring framework encompasses many previously proposed provenance models. This includes where-provenance as soon as we annotate other data elements beyond tuples, as shown in [10] (see also [26]).

Recognizing the limitations of why-provenance, [6] introduces *route-provenance*, which is closest to the provenance model used in ORCHESTRA, albeit used for a different purpose—debugging schema mappings. Our model maintains a graph from which provenance can be incrementally recomputed or explored, whereas in [6] the shortest route-provenances are recomputed on demand. [5] contains many more references on provenance that space does not allow us to mention here.

ORCHESTRA builds upon foundations established by PDMS (e.g., [17]) and data exchange [24, 9]. In [11], the authors use target-to-source tgds to express trust. We support a more flexible trust model where each peer may express different levels of trust for other peers, and trust conditions compose along paths of mappings. Moreover, our approach does not increase the complexity of computing a solution.

7 Conclusions

The ORCHESTRA project represents a re-thinking of how data should be shared at large scale, when differences of opinion arise not only in the data representation, but also which data is correct. It defines new models and algorithms for update exchange, provenance, trust, and more. Our initial prototype system demonstrates the feasibility of the CDSS concept, and we are releasing it into open source at `code.google.com/p/penn-orchestra`. We believe that many opportunities for further research are enabled by our platform. For example, we believe there are many interesting avenues of exploration along derivations, conflicting data, data versions, etc. We also feel it would be worthwhile to explore integrating probabilistic data models into the CDSS architecture.

Acknowledgements. We thank the other members of the ORCHESTRA team, particularly Nicholas Taylor, Olivier Biton, and Sam Donnelly, for their contributions to the effort; and the Penn Database Group and Wang-Chiew Tan for their feedback. This work was funded in part by NSF IIS-0447972, IIS-0513778, IIS-0713267, and CNS-0721541, and DARPA HRO1107-1-0029.

References

[1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *PODS*, Seattle, WA, 1998.

[2] C. Beeri and M. Vardi. A proof procedure for data dependencies. *JACM*, 31(4), October 1984.

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

[4] P. Buneman, S. Khanna, and W.-C. Tan. Why and where: A characterization of data provenance. In *ICDT*, 2001.

[5] J. Cheney, L. Chiticariu, and W.-C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4), 2009.

[6] L. Chiticariu and W.-C. Tan. Debugging schema mappings with routes. In *VLDB*, 2006.

[7] Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM TODS*, 25(2), 2000.

[8] U. Dayal and P. A. Bernstein. On the correct translation of update operations on relational views. *TODS*, 7(3), 1982.

[9] R. Fagin, P. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. *TCS*, 336, 2005.

[10] J. N. Foster, T. J. Green, and V. Tannen. Annotated XML: Queries and provenance. In *PODS*, 2008.

[11] A. Fuxman, P. G. Kolaitis, R. J. Miller, and W.-C. Tan. Peer data exchange. In *PODS*, 2005.

[12] T. J. Green. Containment of conjunctive queries on annotated relations. In *ICDT*, 2009.

[13] T. J. Green, Z. G. Ives, and V. Tannen. Reconcilable differences. In *ICDT*, 2009.

[14] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update exchange with mappings and provenance. In *VLDB*, 2007.

[15] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, Beijing, China, June 2007.

[16] A. Gupta and I. S. Mumick, editors. *Materialized Views: Techniques, Implementations and Applications*. The MIT Press, 1999.

[17] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *ICDE*, March 2003.

[18] Z. Ives, N. Khandelwal, A. Kapur, and M. Cakir. ORCHESTRA: Rapid, collaborative sharing of dynamic data. In *CIDR*, January 2005.

[19] G. Karvounarakis. *Provenance in Collaborative Data Sharing*. PhD thesis, University of Pennsylvania, 2009.

[20] G. Karvounarakis and Z. G. Ives. Bidirectional mappings for data and update exchange. In *WebDB*, 2008.

[21] G. Karvounarakis, Z. G. Ives, and V. Tannen. Querying data provenance. In *SIGMOD*, 2010.

[22] L. Kot and C. Koch. Cooperative update exchange in the Youtopia system. In *Proc. VLDB*, 2009.

[23] P. Mork, R. Shaker, A. Halevy, and P. Tarczy-Hornoch. PQL: A declarative query language over dynamic biological schemata. In *American Medical Informatics Association (AMIA) Symposium, 2002*, November 2002.

[24] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating web data. In *VLDB*, 2002.

[25] P. P. Talukdar, M. Jacob, M. S. Mehmood, K. Crammer, Z. Ives, F. Pereira, and S. Guha. Learning to create data-integrating queries. In *VLDB*, 2008.

[26] V. Tannen. Provenance for database transformations. In *EDBT*, 2010. Keynote talk; slides available on the author's homepage.

[27] N. E. Taylor and Z. G. Ives. Reconciling while tolerating disagreement in collaborative data sharing. In *SIGMOD*, 2006.

[28] N. E. Taylor and Z. G. Ives. Reliable storage and querying for collaborative data sharing systems. ICDE, 2010.