Reexamining Some Holy Grails of Data Provenance

Boris Glavic
University of Toronto

Renée J. Miller
University of Toronto

Abstract

We reconsider some of the explicit and implicit properties that underlie well-established definitions of data provenance semantics. Previous work on comparing provenance semantics has mostly focused on *expressive power* (does the provenance generated by a certain semantics subsume the provenance generated by other semantics) and on understanding whether a semantics is *insensitive to query rewrite* (i.e., do equivalent queries have the same provenance). In contrast, we try to investigate why certain semantics possess specific properties (like insensitivity) and whether these properties are always desirable. We present a new property *stability with respect to query language extension* that, to the best of our knowledge, has not been isolated and studied on its own.

1 Introduction

Database Provenance is an area that has been studied intensively in the recent years leading to several well-understood provenance semantics. Previous work [4, 14, 10, 23] has compared the expressive power and established some properties of these semantics. In this work, we take a second look at these results from a different angle and study an additional property. This allows us to gain interesting insights about provenance semantics and to highlight what we believe are the causes of some of their weaknesses. We limit the discussion to the following (naturally incomplete) set of provenance semantics: *Where*-, *Why*-, and *How*-provenance [4], semantics related to *Lineage* [9] including the semantics used by the Perm system [11] (*PI-CS* and *C-CS*), and *Causality* [21]. An overview of these semantics is presented after the discussion of the properties we investigate in this work. To have a uniform representation of provenance we follow the approach from [4] and use the notation $X(Q,I,t)$ to denote the provenance of tuple t from the result of evaluating query Q over instance I according to the provenance semantics X (omitting the instance if it is clear from the context). For a database instance I let $Tuple(I)$ denote the set of all (R,t) relation-tuple pairs from I and $Attr(I)$ all (R,t,A) relation-tuple-attribute pairs from I (leaving out R if it is clear from the context). Let $\mathscr{P}(X)$ denote the power set of set X.

2 What are Desirable Properties For Provenance Semantics?

The properties of provenance semantics that have most dominantly been addressed in the literature are *expressive power* and *insensitivity to query rewrite*. We now review insensitivity, and introduce an additional relevant and natural property of provenance semantics, called stability, that has not been studied in depth.

Insensitivity to Query Rewrite A provenance semantics is insensitive to query rewrite (or short *insensitive*) if under this semantics equivalent queries have the same provenance. This property has been studied extensively in the past and for most of the semantics considered in this paper it is known whether they are insensitive or not. Aside from restating these results, we will try to investigate why semantics possess this property, discuss the different approaches to achieve insensitivity, and argue whether this property is desirable or not. Note that some semantics are only insensitive under set or under bag semantics, because the classes of equivalent queries for set and bag semantics are different.

Stability with Respect to Query Language Extension A provenance semantics is "stable with respect to query language extension" (or short *stable*) if extending the query language it is defined for with new operators does not affect the provenance of queries that do not use the new operators.

Table 1 shows a summary of the properties for each provenance semantics considered in this paper (note that

Semantics		Insensitive (set semantics)	Stable
Why	Wit	X	X
	Why	-	X
	IWhy	X	X
Where	Where	-	X
	IWhere	X	-
How		-	X
Lineage-based	Lineage	-	X
	PI-CS	-	X
	Copy-CS	-	X
Causality		X	X

Table 1: Properties of Provenance Semantics

Representation	Used by
$\mathscr{P}(Attr(I))$	Where, IWhere
$\mathscr{P}(\mathscr{P}(Tuple(I)))$	Wit, Why, IWhy
$\mathbb{N}[Tuple(I)]$	How
$\{< R_1^*, \ldots, R_n^* > \mid R_i^* \subseteq R_i(Q)\}$	Lineage
$\mathscr{P}(\{< t_1, \ldots, t_n > \mid t_i \in R_i(Q) \vee t_i = \perp\})$	PI-CS, C-CS
$\mathscr{P}(Tuple(I))$	Causality

Table 2: ProvenanceRepresentations

for insensitivity we consider set semantics). In the following, we first give an overview of these semantics, and afterwards discuss the properties in more depth and investigate why certain semantics do or do not possess a property.

3 Provenance Semantics

In this section we introduce the provenance semantics discussed in this work. Table 2 shows how these semantics represent provenance information. Examples for these semantics are given in Figure 1.

Why-provenance (Wit, Why, IWhy) [4, 3] Why-provenance models provenance as a set of so-called witnesses. A witness w for a tuple t from the result of $Q(I)$ is a subset of the instance I where $t \in Q(w)$, i.e., each witness is a set of tuples that is sufficient to derive t through Q. Using the $Tuple(I)$ notation, Why-provenance is an element from $\mathscr{P}(\mathscr{P}(Tuple(I)))$. We are considering three variants of this semantics. The *Set of Witnesses* (*Wit*) contains all witnesses of tuple t. This semantics usually includes a large number of irrelevant tuples in the provenance. E.g., I is a trivial witness for every monotone query Q and each super set of a witness is also a witness (as long as it is a subset of I). The *Witness Basis* (*Why*) is a restriction of Wit to so-called *proof-witnesses*. A proof-witnesses contains one tuple from each leaf of the algebra tree for query Q. This means that the witness basis is defined over the syntactic structure of a query. For instance, for a query $R \bowtie S$ each proof-witness contains a tuple from relation R and one from relation S. The third variant of Why provenance, the *minimal witness basis* (*IWhy*), is the set of minimal witnesses for a tuple t. A witness w is minimal, if no subset of w is also a witness. As has been pointed out before [4], IWhy can be derived from both Wit and Why.

Example 1 *For example, $Wit(Q_a, a_1)$ as shown in Figure 1 is the set of all subsets of I that contain tuple r_1. $Why(Q_a, a_1)$ and $IWhy(Q_a, a_1)$ both contain only a single witness: $\{r_1\}$.*

How-provenance [17, 15, 1, 10, 18, 16, 19] Green et al. [17] introduced a provenance semantics that models provenance by annotating tuples with elements from a semi-ring. We only consider the most general form of annotations - polynomials over variables representing tuples from the instance I. Following common terminology we refer to this model as *How* provenance. Let $\mathbb{N}[Tuple(I)]$ denote the set of these polynomials over variables from $Tuple(I)$. Green et al. [17] presented how annotations are propagated for operators of the positive relational algebra. If every tuple in the database instance carries its variable as an annotation, then the result tuples of a query will be annotated with a polynomial that describes how tuples from the instance were combined by the query to derive the output tuples. In a polynomial, addition ($+$) represents disjunctive use of tuples and multiplication ($\times$) indicates conjunctive use.

Example 2 *For instance, consider $How(Q_b, a_1)$ shown in Figure 1. Tuple a_1 is generated by joining tuple r_1 with itself (r_1^2) and by joining r_1 with r_2 ($r_1 \times r_2$).*

Using different semirings, the annotation propagation for semiring annotated relations (called K-relations) defined by Green et al. [17] correctly models several extensions of the relational model such as bag semantics or uncertainty.

Where-provenance [4, 3, 2, 5, 23] The semantics presented above explain which tuples are responsible for the existence of a tuple in a query result. In contrast, Where-provenance (*Where*) explains from which attribute values in the instance the attribute values of a query result tuple have been copied. The provenance of a result attribute value is modeled as a set of instance attribute values (an element of $\mathscr{P}(Attr(I))$). The sensitivity to query

Queries

$Q_a = R$

$Q_b = \Pi_{A,B}(R \bowtie_{A=C} \Pi_{A \to C, B \to D}(R))$

$Q_c = \Pi_A(R) \cup \Pi_B(R)$

$Q_d = \Pi_A(R \bowtie_{B=C} S)$

$Q_e = \Pi_A(S \bowtie_{B=C} R)$

Database Instance And Query Results

R

	A	B
r_1	1	2
r_2	1	3
r_3	2	3
r_4	2	5

S

	C
s_1	2
s_2	3

Q_a & Q_b

	A	B
a_1	1	2
a_2	1	3
a_3	2	3
a_4	2	5

Q_c

	A
c_1	1
c_2	2
c_3	3
c_4	5

Q_d & Q_e

	A
d_1	1
d_2	2

Provenance

$Wit(Q_a,a_1) = Wit(Q_b,a_1) = \{J \mid J \subset R \wedge r_1 \in J\}$

$Why(Q_a,a_1) = \{\{r_1\}\}$

$Why(Q_b,a_1) = \{\{r_1\},\{r_1,r_2\}\}$

$IWhy(Q_a,a_1) = IWhy(Q_b,a_1) = \{\{r_1\}\}$

$PI\text{-}CS(Q_c,c_2) = \{<\perp,r_1>,<r_3,\perp>,<r_4,\perp>\}$

$PI\text{-}CS(Q_d,d_1) = \{<r_1,s_1>,<r_2,s_2>\}$

$PI\text{-}CS(Q_e,d_1) = \{<s_1,r_1>,<s_2,r_2>\}$

$C\text{-}CS(Q_d,d_1) = \{<r_1,\perp>,<r_2,\perp>\}$

$Causality(Q_a,a_1) = Causality(Q_b,a_1) = \{r_1\}$

$Where(Q_a,a_1,A) = \{(r_1,A)\}$

$Where(Q_b,a_1,A) = \{(r_1,A),(r_2,A)\}$

$IWhere(Q_a,a_1,A) = IWhere(Q_b,a_1,A) = \{(r_1,A),(r_2,A)\}$

$IWhere(Q_a,a_3,A) = \{(r_3,A),(r_4,A)\}$

$UWhere(Q_a,a_3,A) = \{(r_3,A),(r_4,A),(r_1,B),(s_1,C)\}$

$How(Q_a,a_1) = r_1$

$How(Q_b,a_1) = r_1^2 + r_1 \times r_2$

$Lineage(Q_d,d_1) = <\{r_1,r_2\},\{s_1,s_2\}>$

$Lineage(Q_e,d_1) = <\{s_1,s_2\},\{r_1,r_2\}>$

Figure 1: Provenance Semantics Examples

rewrite of Where lead to the development an insensitive version of this semantics (which we refer to as *IWhere*). $IWhere(Q,t,A)$ is the union over $Where(Q',t,A)$ for all queries Q' that are equivalent to query Q.

Example 3 *For example, $Where(Q_a,a_1,A)$ (shown in Figure 1) is the singleton set $\{(r_1,A)\}$, because attribute value a_1.A has been copied from this location. For query Q_b that is equivalent to query Q_a, $Where(Q_b,a_1,A)$ contains the additional attribute value (r_2,A), because of the join on attribute A.*

Lineage-based (Lineage, PI-CS, C-CS) [9, 7, 8, 6, 11, 12, 13] Cui et al. [9] introduced one of the first provenance semantics (which we refer to as *Lineage*). This semantics represents provenance as a list of subsets of the relations accessed by a query (appearing in the same order as they are accessed by the algebra tree of the query). Lineage considers tuples to belong to the provenance if they "contributed" to an output tuple (similar to How and Why).

Example 4 *For example, consider $Lineage(Q_d,d_1)$ (Figure 1). The result tuple d_1 is derived by joining tuple r_1 with s_1 and r_2 with s_2. Thus, $Lineage(Q_d,d_1)$ contains the two sets $\{r_1,r_2\}$ and $\{s_1,s_2\}$.*

Glavic et al. [11] presented provenance semantics that are based on Lineage, but use a different provenance representation and are defined for a larger class of queries. This line of work represents provenance as sets of so-called *witness lists*. As shown in Table 2, a witness-list is a list of instance tuples that were used together to derive an output tuple. A witness list for a query Q contains one tuple per relation accessed by Q ($R_i(q)$ denotes the relation accessed by the ith leaf of the algebra tree for Q). A special value $\perp$ is used to indicate that no tuple from an instance relation access belongs to a witness list. Here we present two of the provenance semantics supported by the *Perm* [11] provenance management system. Perm-Influence contribution semantics (*PI-CS*) considers tuples to belong to the provenance if they were used to derive a query result tuple t.

Example 5 *For instance, Figure 1 shows $PI\text{-}CS(Q_c,c_2)$. Tuple c_2 was generated by (1) projecting tuple r_1 on attribute B (the right input of the union) resulting in the witness list $<\perp,r_1>$ and (2) by projecting tuples r_3 and r_4 on attribute A explained by witness lists $<r_3,\perp>$ and $<r_4,\perp>$.*

The second provenance semantics of Perm we are discussing is Copy contribution semantics (*C-CS*). Similar to Where, this semantics studies where data in the query result has been copied from. C-CS uses the same witness list representation of provenance as PI-CS. In contrast to Where, this semantics traces copying on a per tuple basis. Under C-CS a tuple belongs to the provenance of a query result tuple t if attribute values have been copied from this tuple to tuple t.

Example 6 *For instance, $C\text{-}CS(Q_d,d_1)$ is $\{<r_1,\perp>\}$, because the value of attribute A in tuple d_1 has been*

copied from tuple r_1. Tuple r_1 has been joined with tuple s_1 to produce d_1, but no values have been copied from s_1 to the result and, therefore, s_1 does not belong to the provenance.

Causality [22, 21, 20] Based on work from the AI community, Meliou et al. [21] presented a provenance semantics that considers the set of *causes* to be the provenance of a tuple t. Causes are defined declaratively as follows: An instance tuple t_i is a cause for a result tuple $t \in Q(I)$ if (1) $t \notin Q(I - \{t_i\})$ (in this case t_i is called a counterfactual cause) or (2) if there exists a subset C of the instance called *contingency* so that $t \in Q(I - C)$ and $t \notin Q(I - (C \cup \{t_i\}))$.

Example 7 *For instance, consider Causality(Q_a, a_1) shown in Figure 1. Tuple r_1 is a counterfactual cause for a_1, because removing this tuple from the instance causes a_1 to disappear from the result of query Q_a.*

4 Insensitivity to Query Rewrite

Out of the presented provenance semantics only Wit, IWhy, IWhere, and Causality are insensitive. Counterexamples and proofs of insensitivity have been given in the literature [4, 2, 3] (except for PI-CS and C-CS for which simple counterexamples exist and Causality which is naturally insensitive). Here we focus on the question of how these semantics achieve insensitivity and what causes other semantics to be sensitive. To some extent this question was answered before [3, 2, 4] using the argument that semantics that are defined over the syntactic structure of a query are susceptible to be sensitive. We would like to go further and discuss whether insensitivity is a desirable property at all and investigate some of its undesirable side-effects.

Why-provenance Wit is insensitive, because the only condition on witnesses is that they produce a set containing the tuple t for which provenance is generated. Since this is a black-box property of the query (it is stated solely over the input-output behaviour of the query) a semantics that is based only on this property is bound to be insensitive under both set and bag semantics. Why, in spite of being a subset of Wit, is sensitive. Why is constructed over the syntax of a query, but this does not necessarily imply insensitivity. The insensitivity stems from the fact that $Why(Q, I, t)$ may contain tuples that do not "contribute" to the existence of t at all. Thus, equivalent queries that apply redundant computations may have a larger provenance than their non-redundant counterpart. For example, consider the provenance for tuple a_1 from result of example queries Q_a and Q_b (Figure 1). The reason for Why being sensitive is that in addition to being based on the syntactic structure of a query (which implies a potential of insensitivity) it does not take into account if a tuple is needed to derive t. However, as the insensitivity of Wit demonstrates, including irrelevant tuples in the provenance does not necessarily imply sensitivity. In fact, not excluding irrelevant tuples does not affect the insensitivity of Wit, because all super-sets of witnesses are also witnesses. This implies that tuples that may be used by redundant computations are included in the provenance by default. IWhy can be derived from Wit by excluding all non-minimal witnesses from Wit. Using this approach, it is trivial to see that IWhy has to be insensitive: Since Wit is insensitive, $Wit(Q, I, t) = Wit(Q', I, t)$ for equivalent queries Q and Q' and, thus, the minimal witnesses of the two sets have to be the same too.

Where-provenance The example discussed below shows that Where is sensitive. Traditionally, the sensitivity of Where has been attributed to the fact that Where depends on the syntactic structure of a query. But why does this lead to sensitivity? The reason is that the origin of an attribute value in the result of a query depends on how this value is routed through the query, i.e., it depends on the internal data flow of the query.

Example 8 *Consider a standard example for the sensitivity of Where (e.g., used by Cheney et al. [4]). Queries Q_a and Q_b (Figure 1) are equivalent, but their Where provenance for tuple a_1 and attribute A is different. Query Q_a copies the A attribute value from tuple r_1 and Q_b copies the value from tuples r_1 and r_2. Because of the join on attribute A, tuples with the same A attribute value as r_1 are included in the provenance (r_2).*

As mentioned in the last section *IWhere* [2] was developed to deal with the sensitivity of this semantics. IWhere achieves insensitivity by combining the Where provenance for all queries equivalent to a query Q. This approach for achieving insensitivity has the counter-intuitive effect that if the provenance contains an attribute value (R, t, A), then it also contains all attribute values (R, t', A) for which $t.A = t'.A$ (the A attribute values are the same). This leads to strange situations where the provenance contains attribute values from tuples that were not even used to derive a tuple (e.g., value $(r_2.A)$ from the example). The reason for this effect is that if a query Q copies an attribute value from attribute A of source relation R then an equivalent query can be constructed by extending Q with a redundant self-join on A. For example, query Q_b extends Q_a in this way.

How-provenance How is sensitive to query rewrite under set semantics (shown by Cheney et al. [4] using a counterexample). However, this semantics is insensitive under bag semantics as has been shown in the original work that introduced this semantics [17]. Insensitivity stems from the fact that query evaluation on K-relation was defined to take bag semantics equivalences into account.

Example 9 *Consider the how provenance for tuple a_1 from the result of queries Q_a and Q_b (Figure 1). For query Q_a, tuple a_1 has been produced from r_1. For query Q_b the tuple a_1 is in the result because (1) r_1 was joined with itself (r_1^2) and (2) r_1 was joined with r_2 ($r_1 \times r_2$).*

From this example we can abstract that the reason for the sensitivity of How is the same as for Why: including irrelevant tuples in the provenance.

Lineage-based Lineage-based semantics are strongly depended on the syntactic structure of a query. Thus, it is not surprising that Lineage, PI-CS and C-CS are sensitive under both set and bag semantics.

Example 10 *For instance, equivalent queries Q_d and Q_e differ in the order they join relations R and S which causes them to have different Lineage, PI-CS, and C-CS provenance. Consider Lineage and PI-CS for tuple d_1 from the result of these queries as shown in Figure 1. The provenance contains the same tuples from relations R and S, but in different order (the order they are accessed by the query).*

Causality Causality is defined over the black-box behaviour of a query. It follows that this semantics is insensitive under both set and bag semantics.

Example 11 *$Causality(Q_a, a_1)$ and $Causality(Q_b, a_1)$ shown in Figure 1 demonstrate the insensitivity of causality.*

Arguments for Insensitivity Traditionally, insensitivity to query rewrite has been considered as a desirable property for a provenance semantics to possess. The main arguments are: **(1)** query equivalence is a well-established and well-studied field and many advantages of declarative languages stem from knowledge about equivalence classes of queries. For instance, being able to choose different execution plans by applying equivalence preserving rewrites is one of the main enablers of query optimization. Thus, it is not surprising that database researchers tend to believe that invariance under query rewrite (insensitivity) is a desirable property for provenance semantics. **(2)** Given the fact that query optimizers may choose an execution plan that is quite different from how the query was stated by the user, sensitive semantics can lead to paradoxical situations. Usually, provenance is computed for the query as stated by the user, which means it may not reflect the actual execution. **(3)** The last argument is more a practical consideration. If a sensitive semantics is implemented inside a database engine, this would limit the options of the engine to apply optimizations, because not all equivalent execution plans would result in the "correct" provenance being generated.

Discussion Are the arguments for insensitivity presented above meaningful in general? In our opinion, the first argument can be dismissed, because "tradition" is not a solid argument. The paradox mentioned in the second argument can not be prevented for sensible provenance semantics. However, we believe that generating provenance for the query as originally stated by a user is a reasonable approach (for instance, the Perm system [11] applies this policy).

The third argument addresses the practicality of integrating sensible provenance semantics into a database engine. While limiting the optimizers options is certainly undesirable, we believe the argument is weakened by the following observations. (1) Insensitive semantics may be harder to compute resulting in a trade-off between the computational complexity of provenance generation versus limiting the optimizers options. (2) The optimizer search space for queries with provenance computation may be very different from the search space for standard relational queries. Furthermore, integrating provenance computation inside a database execution engine and optimizer has, to the best of our knowledge, not been addressed and, thus, the impact remains to be studied.

In summary, insensitivity has some advantages, but is not a desirable property for all types of provenance semantics. In particular, semantics that address the data-flow inside a query or study how tuples have been combined by a query are not well-suited for insensitivity. Furthermore, the approach to achieve insensitivity should be chosen carefully. For instance, the "maximality" approach chosen for IWhere results in strange side-effects (which we will discuss in the next section).

5 Stability

We would naturally expect most provenance semantics to be stable. Adding new operators to the query language should not affect the provenance of queries that do not use these operators. IWhere is a notable exception to this rule. Bhagwat et al. [2] presented two versions of IWhere, one for SPJ-queries and one for SPJU-queries (we refer to the latter as *UWhere*). This is necessary, because of how insensitivity is achieved by this semantics. Recall that the IWhere provenance for a tuple t from the result of a query Q is the union of the Where provenance for all queries equivalent to Q. Adding a new operator, such as set union, to the query language obviously changes the set of queries that are equivalent to a query Q. The Where provenance of these additional queries may contain additional elements not found in the provenance of the original set of equivalent queries. Thus, UWhere containing these additional elements is a superset of IWhere, i.e., adding the union operator changes the provenance of queries that do not even use this operator.

Example 12 *Consider IWhere and UWhere for the*

value of attribute A from tuple $a_3 \in Q_a$ (Figure 1). The UWhere provenance contains attribute values of tuples from relation S simply because query $R \cup (R \bowtie_{A=C} S)$ is equivalent to Q_a and for this query the A attribute value of a_3 has been copied from tuples from relation S.

As evident in the example, the UWhere semantics considers all attribute values that are equal to an output attribute value A to belong to the provenance of A. In addition to the computational effort needed to compute this kind of semantics, this behaviour is also counterintuitive. This example strengthens our argument that the approach to achieve insensitivity should be chosen carefully to avoid undesirable behaviour such as instability.

6 Conclusions

In this work, we reexamined well-known properties of provenance semantics by reviewing previous results from a different point of view and presented a property that is natural but has not been considered before. Especially, we were interested in examining why a provenance semantics possesses a certain property or not. We hope that the insights we presented in this work will inspire discussion about provenance semantics by the community, and that showing why semantics possess desirable or undesirable properties will aid in the development of future provenance semantics.

References

[1] AMSTERDAMER, Y., DEUTCH, D., AND TANNEN, V. Provenance for Aggregate Queries. In *PODS* (2011), p. to appear.

[2] BHAGWAT, D., CHITICARIU, L., TAN, W., AND VIJAYVARGIYA, G. An Annotation Management System for Relational Databases. *VLDB Journal 14*, 4 (2005), 373–396.

[3] BUNEMAN, P., KHANNA, S., AND TAN, W.-C. Why and Where: A Characterization of Data Provenance. In *ICDT* (2001), pp. 316–330.

[4] CHENEY, J., CHITICARIU, L., AND TAN, W.-C. Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases 1*, 4 (2009), 379–474.

[5] CHITICARIU, L., TAN, W.-C., AND VIJAYVARGIYA, G. DBNotes: a Post-it System for Relational Databases based on Provenance. In *SIGMOD* (2005), pp. 942–944.

[6] CUI, Y. *Lineage Tracing in Data Warehouses*. PhD thesis, Stanford University, 2002.

[7] CUI, Y., AND WIDOM, J. Lineage Tracing for General Data Warehouse Transformations. In *VLDB* (2001), pp. 471–480.

[8] CUI, Y., AND WIDOM, J. Run-time Translation of View Tuple Deletions using Data Lineage. Tech. rep., Stanford University, 2001.

[9] CUI, Y., WIDOM, J., AND WIENER, J. L. Tracing the Lineage of View Data in a Warehousing Environment. *TODS 25*, 2 (2000), 179–227.

[10] GEERTS, F., AND POGGI, A. On Database Query Languages for K-relations. *Journal of Applied Logic 8*, 2 (2010), 173–185.

[11] GLAVIC, B. *Perm: Efficient Provenance Support for Relational Databases*. PhD thesis, University of Zurich, 2010.

[12] GLAVIC, B., AND ALONSO, G. Perm: Processing Provenance and Data on the same Data Model through Query Rewriting. In *ICDE* (2009), pp. 174–185.

[13] GLAVIC, B., AND ALONSO, G. Provenance for Nested Subqueries. In *EDBT* (2009), pp. 982–993.

[14] GREEN, T. J. Containment of Conjunctive Queries on Annotated Relations. In *ICDT* (2009), pp. 296–309.

[15] GREEN, T. J., IVES, Z. G., AND TANNEN, V. Reconcilable Differences. In *ICDT* (2009), pp. 212–224.

[16] GREEN, T. J., KARVOUNARAKIS, G., IVES, Z. G., AND TANNEN, V. Update Exchange with Mappings and Provenance. In *VLDB* (2007), pp. 675–686.

[17] GREEN, T. J., KARVOUNARAKIS, G., AND TANNEN, V. Provenance Semirings. In *PODS* (2007), pp. 31–40.

[18] IVES, Z. G., GREEN, T. J., KARVOUNARAKIS, G., TAYLOR, N. E., TANNEN, V., TALUKDAR, P. P., JACOB, M., AND PEREIRA, F. The ORCHESTRA Collaborative Data Sharing System. *SIGMOD Record 37*, 2 (2008), 26–32.

[19] KARVOUNARAKIS, G., IVES, Z., AND TANNEN, V. Querying Data Provenance. In *SIGMOD* (2010), pp. 951–962.

[20] MELIOU, A., GATTERBAUER, W., HALPERN, J., KOCH, C., MOORE, K., AND SUCIU, D. Causality in Databases. *IEEE Data Engineering Bulletin* (2010).

[21] MELIOU, A., GATTERBAUER, W., MOORE, K., AND SUCIU, D. The Complexity of Causality and Responsibility for Query Answers and non-Answers. *PVLDB 4*, 1 (2010).

[22] MELIOU, A., GATTERBAUER, W., MOORE, K. F., AND SUCIU, D. Why so? or Why no? Functional Causality for Explaining Query Answers. Tech. rep., University of Washington, 2009.

[23] TAN, W.-C. Containment of Relational Queries with Annotation Propagation. *DBPL* (2003).