Data Citation: A New Provenance Challenge

Abdussalam Alawini, Susan Davidson, Gianmaria Silvello, Val Tannen, Yinjun Wu

Abstract

In today's era of big data-driven science, an increasing amount of information is being published as curated online databases and retrieved by queries, raising the question of how query results should be cited. Because it is infeasible to associate citation information with every possible query, one approach is to specify citations for a small set of frequent queries – citation views *– and then use these views to construct a citation for general queries. In this paper, we describe this model of citation views, how they are used to construct citations for general queries, and an efficient approach to implementing this model. We also show the connection between data citation and data provenance.*

1 Introduction

Citation is an essential part of scientific and scholarly publishing. It is crucial for gauging the trust placed in published data and giving appropriate credit to authors. However, the nature of publication is shifting from traditional venues – such as books, journals and conferences – for which citation is well understood, to databases containing curated information which is retrieved by queries. This is especially true in Big Data-driven science, where many scientific reference works and collections of experimental results are now being published as curated on-line databases with web-page views.

Typically, database owners specify the citation to a web-page view as a journal article whose title includes the name of the database and whose author list includes the chief personnel (e.g. the PI, DBA, lead annotator, etc), along with the query and date of access. However, in many cases the content of the query result is contributed by members of the community and curated by experts, who are not on the author list of the journal article, and the lack of appropriate citation is becoming a stumbling block as evidenced by the recent "data parasite" controversy [14]. Appropriate data citation is therefore essential to motivate members of the scientific community to continue to share data and experimental results so that they can be used and built on by others in the advancement of science.

Many of the "citable" databases that we have examined - e.g. the *Reactome Pathway* database [13] and the *eagle-i* [17] resource discovery tool - describe in English what snippets of information are to be included in a citation for data displayed on a web page; however, users must then construct the citation by hand. However, the English specifications are fairly complex, and the effort required to pull the data off the web page discourages users from generating the citations. Users are therefore unlikely to cite this data correctly unless the citations are *automatically* generated and returned to users along with the data retrieved by a query.

The most advanced database from the perspective of citation that we have examined is the *IUPHAR/BPS Guide to Pharmacology (GtoPdb)* [16], a relational database that contains expertly curated information about

Copyright 2018 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

drugs, the cellular targets of the drugs, and their mechanisms of action in the body. In this database, users view information through a hierarchy of web pages. The top level divides information by "families" of drug targets that reflect typical pharmacological thinking; lower levels divide the families hierarchically into sub-families and so on down to individual drug targets and drugs. For data displayed in Family and Family Introduction web pages, GtoPdb places a human readable citation calculated from information in the underlying database in the page; the citation can then be copy-pasted and reformatted into whatever style the user wishes. The citation varies depending on the part of the database being queried (e.g. the particular Family), and contains an identifier for the data along with information about the contents (analogous to a title), the contributors and/or curators of the data (analogous to authors), the date on which the contents were last modified, and the date the database was queried. In the future, owners of GtoPdb would like to enable general queries against the underlying database rather than restricting access to the database to queries expressed as web-pages, and automatically return citations along with the data. The question is: how should citations to general queries be generated?

Data citation is a challenge because, unlike traditional publications which have a fixed granularity to which citations can be attached, the granularity of reference varies; there are a large number of possible queries over a database, each returning a different subset of data. The "snippets" of information to include in the citation may also vary from query to query, e.g. descriptive information about the data subset being returned, analogous to the distinct titles that different chapters have in an edited collection. Note that these snippets of information play an important *human* role: While the query and date of access (or some form of digital object identifier) is sufficient to locate the query result, they are not informative enough if used as a citation as they do not give intuition about the content. This is analogous to the fact that, in traditional journals, a citation like "Nature, 171,737-738" specifies how to locate the article but doesn't tell you why you might want to do so, whereas adding the information "Watson and Crick: Molecular Structure of Nucleic Acids" does. It is therefore necessary to be able to *specify* citations to query results [5].

Since it is impossible to specify the content of a citation to every possible query over a database, one strategy is to specify citations to a small number of frequent queries – *citation views* [8, 9] – and use these to construct citations to other "general" queries. The citation views may be combined (*jointly* used) to construct the citation, and there may be *alternate* ways in which combinations of citation views can be used. The interpretations of joint and alternate use (e.g. union or join) are *policies* to be specified by the database owner.

There is an interesting connection between data citation and data provenance: Naively, citation captures the "origins" of data by giving credit to the people responsible for it. However, the connection goes deeper by viewing both citation and provenance as *annotations on data that are carried through queries*. That is, each tuple t in a base relation is annotated with a view iff t is used to construct the materialized view. When a user query is issued, the view annotations are regarded as provenance tokens to be propagated along with values to the final query result. The view annotations of each tuple are then reasoned over to determine whether or not they are valid in a citation for the tuple.

In the remainder of this paper we explore this connection. We start in Section 2 by describing the model of citation views and how they are used to construct citations to general queries. In Section 3 we discuss the connection to *where*- and *why*-provenance [6, 11]. We then describe in Section 4 an efficient approach to implementing the model which draws on the ideas of the previous two sections. We conclude in Section 5.

2 Model of Citation Views

The citation framework is based on *conjunctive queries* [1]. Conjunctive queries are "universal" across different types of databases (e.g. relational, semistructured, RDF, etc.), and simplify the reasoning used to generate citations. Throughout the paper, we will use Datalog as the syntax for queries, and assume that fresh variables are introduced everywhere in relational subgoals rather than being reused.

We start by defining the notion of citation views, which defines how to associate citations to a fixed set of

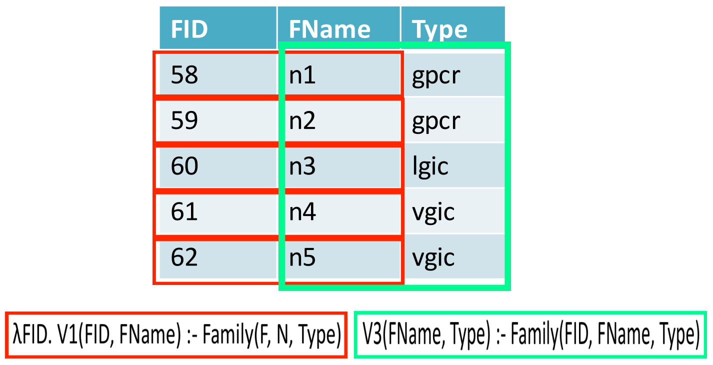
FID
FName
Type
58
n1
gpcr
59
n2
gpcr
60
n3
lgic
61
n4
vgic
62
n5
vgic
λFID. V1(FID, FName) :- Family(F, N, Type)
V3(FName, Type) :- Family(FID, FName, Type)

Figure 1: Effect of Parameters on Views

frequent queries, and then give a semantics for how to associate citations to *general* queries based on citation views.

2.1 Citation views.

A citation view specifies: 1) the data being cited (*view definition*); 2) the information to be used to construct the citation (*citation queries*); and 3) how the information is combined to construct the citation (*citation function*). The citation function takes the snippets of information retrieved from citation queries as input, and generates an appropriately formatted citation as output (e.g. human readable, BibTex, RIS or XML). Note that the snippets of information required for the citation must be in the database, and that the citation can be thought of as an *annotation* on every tuple in the view result.

The view definition and citation queries are optionally *parameterized*, where the parameters (λ-*variables*) appear as variables somewhere in the body of the query. [1] A parameterized view creates a set of *instantiated views*, one for each possible choice of parameters. The number of such views is therefore instance-dependent, as illustrated in Figure 1. We will use the input parameter(s) to distinguish such views, e.g. V1(60) refers to the instantiated view V1 for FID=60.

Example 1: Recall that in the GtoPdb database, users view information through a hierarchy of web pages. The top level divides information by families of drug targets; lower levels divide the families into sub-families and so on down to individual drug targets and drugs. The content of a particular family "landing" page (referred to as the Family relation) is curated by a committee of experts; a family may also have a "detailed introduction page" (referred to as the FamilyIntro relation) which is written by a set of contributors, who are not necessarily the same as the committee of experts for the family.

Citation views for the Family and FamilyIntro relations can be specified as follows. Views V1, V2 and V4 are parameterized by the key FID, whereas V3 is unparameterized.

$$\begin{array}{ll} \lambda FID.V1(FID, FName) & :-Family(FID, FName, Type) \\ \lambda FID.V2(FID, Text) & :-FamilyIntro(FID, Text) \\ \qquad V3(FName, Type) & :-Family(FID, FName, Type) \\ \lambda FID.V4(FID, FName, Text) & :-Family(FID, FName, Type), FamilyIntro(FID1, Text), FID = FID1 \end{array}$$

For each view V, we define one or more citation queries C_V. We show below examples of citation queries for views V1 and V3, where relation FC captures the committee members who curate the content for V1 and relation MetaData captures general snippets of information such as the owner and url of the database.

$$\begin{array}{ll} \lambda FID.\, C_{V1}(FID, FName, PName) & :-\ Family(FID, FName, Type), FC(FID, PID), Person(PID, PName) \\ \qquad C_{V3}(X1, X2) & :-\ MetaData(T1, X1), T1 = \text{‘}Owner\text{'}, MetaData(T2, X2), T2 = \text{‘}URL\text{’} \end{array}$$

[1] Also called *binding patterns* in [15].

This information can then be used to construct citations for tuples in the query result. For example, the JSON-formatted citation for V1(60) could be {FID: '60', FName: 'RXFP', PName: ['Roger', 'Ross']}, and that for V3 could be {Owner: ['Alexander', 'Davenport'], URL: 'http://www.guidetopharmacology.org'}. □

Attaching citations to general queries. To give a semantics to citations for *general queries*, we use the following intuition: *If a view tuple can be used to create a tuple and is visible in the query result, then the result tuple carries the view tuple's citation annotation.* To do this, mappings between the view definitions and input query are created which maximally and non-redundantly cover the query subgoals; we call this a *covering set* of mappings [3]. This is similar to the notion of query rewriting using views as used, for example, in query optimization and data integration [12]. More formally:

Definition 1: Given a view definition V and query Q

$$\begin{aligned}\mathtt{V}(\bar{\mathtt{Y}}) &: -\mathtt{A}_1(\bar{\mathtt{Y}}_1), \mathtt{A}_2(\bar{\mathtt{Y}}_2), \ldots, \mathtt{A}_k(\bar{\mathtt{Y}}_k), \mathtt{condition}(\mathtt{V})\\ \mathtt{Q}(\bar{\mathtt{X}}) &: -\mathtt{B}_1(\bar{\mathtt{X}}_1), \mathtt{B}_2(\bar{\mathtt{X}}_2), \ldots, \mathtt{B}_m(\bar{\mathtt{X}}_m), \mathtt{condition}(\mathtt{Q})\end{aligned}$$

in which A_i, B_j are relational subgoals, and `condition()` are *non-relational* subgoals which include comparisons of variables to constants (called *local* predicates) and comparisons of variables with variables (called *global* predicates if the variables come from different relational subgoals and *local* otherwise). Then a **view mapping** M from V to Q is a tuple (h, ϕ) in which:

- h is a partial one-to-one function from $\{A_1, ..., A_k\}$ to $\{B_1, ..., B_m\}$ which 1) maps A_i to B_j only if they have the same relation name; and 2) cannot be extended to include more subgoals of Q (i.e. there is no unmapped A_i, B_j which have the same relation name).
- ϕ are the variable mappings from $\bar{Y}' = \cup_{i=1}^{k} \bar{Y}_i$ to $\bar{X}' = \cup_{i=1}^{m} \bar{X}_i$ induced by h

A relational subgoal B_j of Q is *covered* iff $h(A_i) = B_j$ for some i. A variable $x_j \in \bar{X}'$ is *covered* iff $\phi(y_i) = x_j$ for some variable $y_i \in \bar{Y}'$. Note that a view may be in zero or more view mappings for a given query.

We also use the notion of the *extension* of Q, called Q_{ext}, which expands the head of Q to include all variables in the body ($\bar{X}'$).

Definition 2: Valid View Mapping Given a database instance D, a view mapping $M = (h, \phi)$ of V is *valid* for a tuple $t \in Q_{ext}(D)$ iff:

- The projection of t on the variables that are mapped in Q_{ext} under the mapping ϕ is a tuple in $V_{ext}(D)$: $\Pi_{\phi(\bar{Y}')} t \in V_{ext}(D)$
- There exists at least one variable $y \in \bar{Y}$ such that $\phi(y)$ is a distinguished variable.
- All lambda variables in V are mapped to variables in $\bar{X}'$.

Given a set of views $\mathcal{V}$, a query Q and a database instance D, we can build a set of valid view mappings $\mathcal{M}(t)$ for each tuple $t \in Q(D)$ according to Definitions 1 and 2. We then combine different view mappings from $\mathcal{M}(t)$ to create a *covering set* of views for t.

Definition 3: Covering set Let $C \subseteq \mathcal{M}(t)$ be a set of valid view mappings. Then C is a covering set of view mappings for t iff it is *maximal* and *nonredundant*:

- No $V \in \mathcal{M}(t) \setminus C$ can be added to C to cover more subgoals of Q or variables in $\bar{X}$; and

- No $V \in C$ can be removed from C and cover the same subgoals of Q and variables in $\bar{X}$.

Within a covering set, the citation views are *jointly* used (indicated by "*") to construct a citation, and if there are more than one covering set the citations of the covering sets are *alternatively* used (indicated by "$+^R$") to construct a citation to each tuple. The citations for each tuple in the query result can then be *aggregated* (indicated by *Agg*) to form a citation for the entire query result.

Example 2: In this example, there will be at most one valid view mapping from a view V to a query Q; we will therefore use the name of the view as the name of the mapping.

Suppose we had the input query

$$Q1(Name) : -Family(FID, Name, Type), FID >= 60$$

For this, one covering set is {V1}: there is a mapping from the body of V1 to the relational subgoal (Family) as well as an input to the parameter FID. {V3} and {V4} are also covering sets; note that mapping V4 is partial since FamilyIntro is not mapped to any relational subgoal in $Q1$. The citation for the first tuple (FID=60) would therefore alternatively use the citations for V1(60), V3, and V4(60), i.e. it would be Cite($V1(60) +^R V3 +^R V4(60)$). If "$+^R$" were interpreted as "most specific", the resulting citation would be Cite($V1(60)$). Similarly, for tuple FID=62 the citation would be Cite($V1(62) +^R V3 +^R V4(62)$). The citations for each tuple are then aggregated to derive a citation for the entire query result. If aggregation were interpreted as some form of intersection, and "$+^R$" balanced size with specificity, the citation for the query result would be Cite(V3).

On the other hand, consider the following the input query

$$Q2(Type, Text) : -Family(FID1, Name, Type), FamilyIntro(FID2, Text), FID1 = FID2 = 60$$

One covering set is {V2, V3}: there are mappings from the body of V3 to the relational subgoal Family as well as from the body of V2 to the relational subgoal FamilyIntro. We would therefore jointly use the citations for V3 and V2, written Cite($V3 * V2(60)$). Another covering set is {V1, V2}, resulting in citation Cite($V1(60) * V2(60)$). Finally, there is a mapping from the body of V4 which covers all relational subgoals of $Q2$, however {V4} is not a covering set since it only covers the distinguished variable $Text$, and can be augmented with V3 to cover both $Type$ and $Text$. The third covering set is therefore {V3, V4}. The final citation for the single result tuple is therefore Cite($V1(60) * V2(60) +^R V3 * V2(60) +^R V3 * V4(60)$).

Finally, suppose we had an input query which is a subset of the cross product of Family and FamilyIntro:

$$Q3(Type, Text) : -Family(FID1, Name, Type), FamilyIntro(FID2, Text), FID1 <= 60, FID2 <= 60$$

Note that some of the tuples in the result may be in the join of the two tables, and therefore be visible in V4, while others are not. This motivates the need to evaluate the *extended* query $Q3_{ext}$. Since this returns the value of all variables in the body, the validity of the join predicate FID1=FID2 can be evaluated for each result tuple, thereby determining whether the V4 is valid for that particular tuple. Thus, for tuples in the result that are not in the join, the covering sets would be {V1, V2} and {V3, V2}, while for tuples in the join it would also include {V3, V4} (as in $Q2$). □

3 Connection to Provenance

To understand the connection to provenance, we start with a simple but common subclass of view definitions called *partitioning views* which corresponds to *where-* provenance, and then move to the (more complex) general case which corresponds to *why*-provenance [6, 11].

3.1 Partitioning Views

A set of select-project views over a single relation R is partitioning if each attribute of R appears in at most one view; a set of views is partitioning over a database schema $\mathcal{S}$ if it is partitioning for each $R \in \mathcal{S}$.[2] As an example, the following set of citation views is partitioning for our running example:

$\lambda FID.V10(FID, FName)$:- $Family(FID, FName, Type), Type = \text{'}gpcr\text{'}$
$\lambda FID.V11(Type)$:- $Family(FID, FName, Type)$
$\lambda FID.V12(FID, Text)$:- $FamilyIntro(FID, Text)$

In this case, since fresh variables are introduced for every relational subgoal in queries, each attribute of each tuple in a query result is visible in at most one view, and must be the same view across all tuples.[3]

In *where*-provenance [4, 10], each attribute of each tuple in an instance of a relation is annotated with a (unique) provenance token, which is then carried through queries to annotate tuples in the query result. Assume for now that duplicates in queries are not removed, and that views are *materialized*. Then the citation for each tuple in the query result would be the set (*joint use*) of citation views in which the where-provenance tokens for attributes in the tuple appear. When duplicates are removed, the union of the sets of citation views for each duplicate tuple would be used.

Table 3: Base relations with provenance tokens

(a) Family

	FID		FName		Type		
t_1	58	a_1	n1	a_6	gpcr	a_{11}	s_1
t_2	59	a_2	n2	a_7	gpcr	a_{12}	s_2
t_3	60	a_3	n3	a_8	lgic	a_{13}	s_3
t_4	61	a_4	n4	a_9	vgic	a_{14}	s_4
t_5	62	a_5	n5	a_{10}	vgic	a_{15}	s_5

(b) FamilyIntro

	FID		Text		
t'_1	58	b_1	tx1	b_5	r_1
t'_2	60	b_2	tx2	b_6	r_2
t'_3	61	b_3	tx3	b_7	r_3
t'_4	62	b_4	tx3	b_8	r_4

Table 4: Materialized views V10-V12 and query result Q4(D) with provenance tokens

(a) V10

	FID		FName	
t_{101}	58	a_1	n1	a_6
t_{102}	59	a_2	n2	a_7

(b) V11

	Type	
t_{111}	gpcr	a_{11}, a_{12}
t_{112}	lgic	a_{13}
t_{113}	vgic	a_{14}, a_{15}

(c) V12

	FID		Text	
t_{121}	58	b_1	tx1	b_5
t_{122}	60	b_2	tx2	b_6
t_{123}	61	b_3	tx3	b_7
t_{124}	62	b_4	tx3	b_8

(d) $Q4(D)$

	FID		FName		Type		covering sets
t_{q41}	58	$b_1 \to \{V12\}$	n1	$a_6 \to \{V10\}$	gpcr	$a_{11} \to \{V11\}$	$V12 * V10 * V11$
t_{q42}	60	$b_2 \to \{V12\}$	n3	$a_8 \to \{\}$	lgic	$a_{13} \to \{V11\}$	$V12 * V11$
t_{q43}	61	$b_3 \to \{V12\}$	n4	$a_9 \to \{\}$	vgic	$a_{14} \to \{V11\}$	$V12 * V11$
t_{q44}	62	$b_4 \to \{V12\}$	n5	$a_{10} \to \{\}$	vgic	$a_{15} \to \{V11\}$	$V12 * V11$

Example 3: Consider the provenance-annotated relations in Table 3, and ignore for now the annotations in the last column. Observe that for tuple t_1 in Family the provenance annotation for FID is a_1 and that for FName is

[2]Note that more complicated cases of partitioning could also be considered, e.g. *horizontal* in which conditions on variables in the views are mutually exclusive.

[3]Recall that an attribute appears in at most one view, but that a local predicate may cause some tuples in the underlying relation not to be considered. For example, V10 only applies to tuples in Family whose type is 'gpcr'.

a_6. In the materialized instance of V10 shown in Table 4a, tuple t_{101} is the projection of t_1 and therefore carries these annotations.

Assume we have the following query which joins the Family and FamilyIntro relations:

Q4(FID, FName, Type):- Family(FID1, FName, Type), FamilyIntro(FID, Text), FID1=FID

The annotated result of Q4(D) is shown in Figure 4d, where we show the mapping from each where-provenance token to the materialized view in which it occurs. For example, the provenance token b_1 in the result tuple t_{q41} appears in tuple t_{121} of $V12(D)$. For each tuple in the query result, the combination of views from each where-provenance token are *jointly* combined to cover as many distinguished variables of query as possible. For instance, for tuple t_{q41} the combination of views is $V10 * V11 * V12$, which is the covering set for tuple t_{q41}. □

3.2 General views

However, when views are *not* partitioning (as in the case of views V1-V4), where-provenance is no longer sufficient; we must also understand how the tuple in the result was constructed from the input tuples (*why*-provenance). The final column in Table 3 represents the why-provenance for each tuple.

Example 4: The materialized instances of views V1-V4 from Example 1, together with their provenance annotations, are shown in Tables 5a-5d and the result of query Q3 from Example 2 with provenance annotations is shown in Table 5e. Note that the views V1-V4 and the query Q3 carry the provenance tokens from the underlying relations.

Table 5: Materialized views V1-V4 and Query result Q3(D) with provenance tokens

(a) V1

	FID		FName		
t_{11}	58	a_1	n1	a_6	$\{s_1\}$
t_{12}	59	a_2	n2	a_7	$\{s_2\}$
t_{13}	60	a_3	n3	a_8	$\{s_3\}$
t_{14}	61	a_4	n4	a_9	$\{s_4\}$
t_{15}	62	a_5	n5	a_{10}	$\{s_5\}$

(b) V2

	FID		Text		
t_{21}	58	b_1	tx1	b_5	$\{r_1\}$
t_{22}	60	b_2	tx2	b_6	$\{r_2\}$
t_{23}	61	b_3	tx3	b_7	$\{r_3\}$
t_{24}	62	b_4	tx3	b_8	$\{r_4\}$

(c) V3

	FName		Type		
t_{31}	n1	a_6	gpcr	a_{11}	$\{s_1\}$
t_{32}	n2	a_7	gpcr	a_{12}	$\{s_2\}$
t_{33}	n3	a_8	lgic	a_{13}	$\{s_3\}$
t_{34}	n4	a_9	vgic	a_{14}	$\{s_4\}$
t_{35}	n5	a_{10}	vgic	a_{15}	$\{s_5\}$

(d) V4

	FID		FName		Text		
t_{41}	58	a_1	n1	a_6	tx1	b_5	$\{s_1, r_1\}$
t_{42}	60	a_3	n3	a_8	tx2	b_6	$\{s_3, r_2\}$
t_{43}	61	a_4	n4	a_9	tx3	b_7	$\{s_4, r_3\}$
t_{44}	62	a_5	n5	a_{10}	tx3	b_8	$\{s_5, r_4\}$

(e) $Q3(D)$

	Type		Text		
t_{q31}	gpcr	a_{11}	tx1	b_5	$\{s_1, r_1\}$
t_{q32}	gpcr	a_{12}	tx2	b_6	$\{s_2, r_1\}$
t_{q33}	lgic	a_{13}	tx1	b_5	$\{s_3, r_1\}$
t_{q34}	gpcr	a_{11}	tx2	b_6	$\{s_1, r_2\}$
t_{q35}	gpcr	a_{12}	tx1	b_5	$\{s_2, r_2\}$
t_{q36}	lgic	a_{13}	tx2	b_6	$\{s_3, r_2\}$

The validity of the view mappings must be considered for each tuple in the query result; however, this cannot be determined simply by reasoning over the where-provenance tokens. For example, for tuple $t_{q32} \in Q3(D)$,

Table 6: Query result $Q3(D)$ with provenance annotations and valid views

	Type		Text		why-provenance	covering sets
t_{q31}	gpcr	$a_{11} \rightarrow \{V3\}$	tx1	$b_5 \rightarrow \{V2, V4\}$	$\{s_1, r_1\}$	$V3 * V2 +^R V4 * V3$
t_{q32}	gpcr	$a_{12} \rightarrow \{V3\}$	tx2	$b_6 \rightarrow \{V2\}$	$\{s_2, r_1\}$	$V3 * V2$
t_{q33}	lgic	$a_{13} \rightarrow \{V3\}$	tx1	$b_5 \rightarrow \{V2\}$	$\{s_3, r_1\}$	$V3 * V2$
t_{q34}	gpcr	$a_{11} \rightarrow \{V3\}$	tx2	$b_6 \rightarrow \{V2\}$	$\{s_1, r_2\}$	$V3 * V2$
t_{q35}	gpcr	$a_{12} \rightarrow \{V3\}$	tx1	$b_5 \rightarrow \{V2\}$	$\{s_2, r_2\}$	$V3 * V2$
t_{q36}	lgic	$a_{13} \rightarrow \{V3\}$	tx2	$b_6 \rightarrow \{V2, V4\}$	$\{s_3, r_2\}$	$V3 * V2 +^R V4 * V3$

there are two where-provenance tokens, i.e. a_{12} for attribute "Type" and b_6 for attribute "Text", which come from tuple t_2 in the Family relation and t'_2 from the FamilyIntro relation. There exists a view mapping from view $V4$ to Q2, however its join condition (global predicate) under the mapping, $FID1 = FID2$, is not satisfied since the attribute FID in tuples t_2 and t'_2 do not match. However, if we just compare the where-provenance tokens, the token b_6 exists in both tuple $t_{q32} \in Q3(D)$ and the tuple $t_{42} \in V4(D)$, which leads to the incorrect conclusion that there is a valid view mapping for view $V4$ in tuple t_{q32}.

The validity of view mappings can be checked by reasoning over the result of the extended query, $Q3_{ext}$, as illustrated in Example 2. However, there is an alternative approach which reasons over why-provenance information. For example, the last column in Tables 5a-5e records the why-provenance annotations for each tuple in the materialized views and query result. If we consider tuple t_{q31} in $Q3(D)$ again, the corresponding why-provenance annotation also appears in tuple t_{41} of $V4(D)$, which means that the tuples from Family and FamilyIntro used to construct tuple t_{q31} can also work for the construction of tuple t_{41}. This implies that the join condition under the mapping for $V4$ can be satisfied by tuple t_{q31}, hence that the view mapping is valid for t_{q31}. However, for tuple t_{q32}, the why-provenance annotation is $\{s_2, r_1\}$ which does not exist in any tuple in $V4(D)$ and thus the view mapping of $V4$ is not valid.

After checking the validity of views for each tuple in the query result, a mapping is built between each where-provenance token a of a tuple t and the valid materialized views for t in which the token appears. For example, as Table 6 shows, for tuple t_{q31}, the token a_{11} is mapped to view $V3$ while the token b_5 is mapped to views $V2$ and $V4$. In order to cover the attributes "Type" and "Text", the views from each where-provenance token are combined. Notice that we can combine $V3$ with either $V2$ or $V4$ for tuples t_{q31} and t_{q36}, which leads to two *alternative* combinations. □

4 Implementing the Model

In this section, we describe an implementation of the model presented in Section 2 which generates citations for individual tuples in the query result. The citations can then be aggregated to compute a citation to any subset of the query result. The approach was implemented in a prototype, demonstrated in [3], and is called the Tuple Level Approach (TLA). TLA is similar to the *eager approach* to computing provenance [7], which also uses an extended query to carry an extra annotation column from the database. Note that in the TLA implementation, citation views are *not* materialized and all reasoning is done using the instance returned by the extended query discussed in Section 2.

In TLA, as much initial work is done as possible for reasoning about covering sets: each tuple is annotated with all views in which it potentially participates. This is done by expanding the schema of each base relation R with a single column (called the *view vector* column), and adding view V to the view vector for tuple t in R whenever R appears as a relational subgoal of V and t satisfies the local predicates of V. Checking global predicates in V is delayed until the user query is executed. Sample instances with view vectors for the $Family$ and $FamilyIntro$ relations are shown in Tables 7 and 8.

Table 7: Sample table for base relation Family

Family_id	Name	Type	View vector
58	n1	gpcr	V1,V3,V4
59	n2	gpcr	V1,V3,V4
60	n3	lgic	V1,V3,V4
61	n4	vgic	V1,V3,V4
62	n5	vgic	V1,V3,V4

Table 8: Sample table for base relation FamilyIntro

Family_id	Text	View vector
58	tx1	V2,V4
60	tx2	V2,V4
61	tx3	V2,V4
62	tx3	V2,V4

The approach is implemented in four steps:

Preprocessing step. When a query $Q : -B_Q$ is issued, we first calculate all *possible* view mappings according to Definition 1. View mappings will be filtered out in this step if they cannot cover any distinguished variables of Q (the second condition of Definition 2). This result is a set of view mappings $M(Q)$.

In order to derive valid view mappings for each tuple in the query result, Q is extended to include view vectors of every base relation occurring in B_Q and the boolean expressions of any *global predicates* under the view mappings $M(Q)$. The lambda variables under all view mappings in $M(Q)$ will also be included in the head of the extended query if they are not distinguished variables of Q.

Query execution step. The extended query, Q_{ext1}, is then executed over the database instance D yielding an instance $Q_{ext1}(D)$ over which the citation reasoning occurs.

Reasoning step. In first phase of citation reasoning, valid view mappings within each view vector are calculated for each tuple $t \in Q_{ext1}(D)$. A view mapping M from $M(Q)$ is valid for a view vector from relational subgoal R iff there exists a view annotation V in this view vector so that M can be derived from V, the global predicates under M are satisfied, and M covers at least one head variable in the query. In the second phase, combinations of valid view mappings from each view vector are considered to find the covering sets.

Population step To avoid the expense of calculating covering sets tuple by tuple, subsets of tuples that will *share* the same covering sets are found using the view vectors and boolean values of global predicates. Such tuples are then grouped; covering sets are calculated once per group and propagated to all tuples in the group. For example, in Table 9, the result tuples form a single group and therefore the covering set is only calculated once. This optimization leads to significant performance gains.

Example 5: Consider the following query:

$$Q5(Type, Text) : -Family(FID1, Name, Type), FamilyIntro(FID2, Text), FID1 = FID2$$

After deriving valid view mappings within each view vector, the resulting instance of the extended query $Q5_{ext1}(D)$ is shown in Table 9. It is worth noting that the boolean expression of the global predicate $FID = FID1$ from $V4$ is not evaluated since it matches the global constraint of $Q5$. Since each view in this example only has one view mapping, we use $V1, V2, \ldots, V4$ to denote the corresponding view mappings.[4] Note that $FID1, FID2$ are included in the head of $Q5_{ext1}$ since they are lambda variables. The final query result with covering sets is shown in Table 10. Parameterized views are instantiated by passing the parameter values (e.g. V1(59) indicates V1 for FID=59). Multiple covering sets for each tuple are combined with $+^R$ (alternative use). After projecting over the distinguished variables of $Q5$, the third tuple and the fourth tuple in $Q5_{ext1}(D)$ share

[4]In general, since a query may use the same relation more than once in a subgoal, a view may have multiple view mappings.

Table 9: Result of executing the extended query, $Q5_{ext1}(D)$

Type	Text	FID1	FID2	Valid view mappings from view vector 1	Valid view mappings from view vector 2
gpcr	$tx1$	58	58	V3	V2, V4
gpcr	$tx2$	60	60	V3	V2, V4
vgic	$tx3$	61	61	V3	V2, V4
vgic	$tx3$	62	62	V3	V2, V4

Table 10: The final result, $Q5(D)$, annotated with the covering sets

Type	Text	Covering sets
gpcr	$tx1$	$V3 * V2(58) +^R V4(58) * V3$
lgic	$tx2$	$V3 * V2(60) +^R V4(60) * V3$
vgic	$tx3$	$(V3 * V2(61) +^R V4(61) * V3) + (V3 * V2(62) +^R V4(62) * V3)$

the same $Type$ and $Text$ and thus the covering sets of the two tuples are combined with a new alternate use operator, denoted $+$.

Citations are then generated for each tuple using the covering sets, the citation query and function for each view in the covering sets, and the policies for *, $+^R$, and $+$.

For example, suppose $+^R$ is interpreted as min according to a cost function which calculates the total number of unmatched terms (distinguished variables or subgoals) between the views in a covering sets and the query. Then for each tuple in Table 10 the resulting covering set for each tuple will be $V3 * V2(FID)$. Furthermore, suppose $*, +$ and Agg are interpreted as join, union and intersection, respectively, and that the JSON-formatted citations for view $V3$ and each instantiated view $V2$ are as shown in Table 11. Then the citation for the covering set $V3 * V2(58)$ in the first tuple of the query result would be {ID: '58', author: ['Mark', 'Steve', 'Roger'], Committee: ['Poyner', 'Hay', 'Justo']}, which is the *join* of the citations from $V3$ and $V2(58)$. To construct a citation for the entire query result, the citations from each tuple ($V3 * V2(FID)$) in the query result are aggregated, yielding {ID: ['58', '60', '61', '62'], author: ['Mark', 'Steve', 'Roger', 'Jens', 'Rodrigo', 'John'], Committee: ['Hay', 'Poyner', 'Justo', 'Andrew', 'Leo', 'Joel']}. □

Overview of citation framework. The architecture of the citation framework is shown in Figure 2. The DBA specifies the citation views and policies for how the views are to be used in constructing a citation to a general query. When a user submits a query to the database, view definitions are mapped to it and their associated citations combined according to the policies; a citation is then returned to the author along with the query result. When a Reader later uses a citation to access the cited data, the process of citation generation is reverse engineered (see dashed arrows in Figure 2). The citation is dereferenced, obtaining the original query and the citation views that were used; note that versioning is an important component of the solution but is not discussed in this paper. A specialized version of this architecture was developed for *eagle-i* as a proof-of-concept [2].

Table 11: Citations for sample views

View	Result of citation function
V2(58)	{ID: '58', author: ['Mark'], Committee: ['Hay', 'Poyner']}
V2(60)	{ID: '60', author: ['Jens'], Committee: ['Andrew']}
V2(61)	{ID: '61', author: ['Rodrigo'], Committee: ['Leo']}
V2(61)	{ID: '62', author: ['John'], Committee: ['Joel']}
V3	{author: ['Steve', 'Roger'], Committee: ['Hay', 'Justo']}

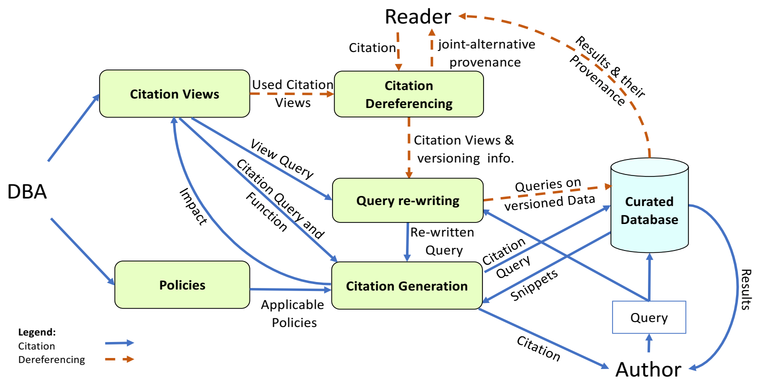

Figure 2: Citation Framework

5 Conclusions

In this paper, we address the problem of generating citations to the results of queries over data published in databases – *data citation* – and explore the connection to *data provenance*, in particular where-provenance and why-provenance. Our approach to data citation is based on *citation views*, as proposed in [9] and implemented in [3]. In this approach, citations are specified to some small number of frequent queries (e.g. web-page views of the database), and are used to construct citations for general queries.

Intuitively, a citation captures the origins of data by giving credit to its authors, i.e. the contributors and/or curators responsible for the data. When the citation views are select-project views over single relations and are partitioning, the view tuples which are used to create a tuple in the result can be simply calculated using where-provenance. However, in the general case where citation views may involve multiple relations (e.g. joins) and may overlap, where-provenance is no longer sufficient; one must also understand how a tuple in the result was constructed from the input tuples (why-provenance).

Rather than constructing and maintaining the materialized citation views as suggested by the connection above, our implementation of citation views reasons solely over the input query Q and the view definitions. It starts by annotating tuples in the base relations with the views in which they potentially participate, and determines which of the potential views are valid for result tuples by evaluating global predicates in $Q_{ext}(D)$. A number of clever optimizations are used to improve the efficiency of the approach, e.g. constructing citations for groups of tuples which share the same covering sets. Initial performance results (not discussed in this paper) show that citations can be generated for typical views and queries in a reasonable amount of time.

In future work, we would like to explore how to *integrate data citation within provenance-enabled* database systems. We would also like to study how existing *versioning* techniques can be adapted for data citation. Note that in this context versioning must be triggered when a user cites a data entry and only needs to record change on the cited data, thus interesting optimizations may be possible. Finally, we would like to explore how citations can be integrated into *data science environments*, in which queries are interleaved with analysis steps. This is difficult since provenance is not well understood in the context of machine learning algorithms.

Acknowledgments. This research was partially funded by NSF IIS 1302212, NSF ACI 1547360, and NIH 3-U01-EB-020954-02S1. The authors would like to thank Peter Buneman for conceiving the idea that data citation is a computational challenge, and Daniel Deutch and Tova Milo for helping formalize the connection to provenance.

References

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

[2] A. Alawini, L. Chen, S. B. Davidson, N. P. D. Silva and G. Silvello. Automating Data Citation: The eagle-i Experience. In *2017 ACM/IEEE Joint Conference on Digital Libraries, JCDL 2017, Toronto, ON, Canada, June 19-23, 2017*, pages 169-178, 2017.

[3] A. Alawini, S. B. Davidson, W. Hu and Y. Wu. Automating Data Citation in CiteDB. *PVLDB*, 10(12):1881-1884, 2017.

[4] P. Buneman, J. Cheney and S. Vansummeren. On the Expressiveness of Implicit Provenance in Query and Update Languages. In *Database Theory - ICDT 2007, 11th International Conference, Barcelona, Spain, January 10-12, 2007, Proceedings*, pages 209-223, 2007.

[5] P. Buneman, S. B. Davidson and J. Frew. Why data citation is a computational problem. *Commun. ACM*, 59(9):50-57, 2016.

[6] P. Buneman, S. Khanna and W. C. Tan. Why and Where: A Characterization of Data Provenance. In *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings.*, pages 316-330, 2001.

[7] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in Databases: Why, How, and Where. *Found. Trends databases*, 1(4):379-474, Apr.2009.

[8] S. B. Davidson, P. Buneman, D. Deutch, T. Milo and G. Silvello. Data Citation: A Computational Challenge. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 1-4, 2017.

[9] S. B. Davidson, D. Deutch, T. Milo and G. Silvello. A Model for Fine-Grained Data Citation. In *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*, http://cidrdb.org, 2017.

[10] J. N. Foster, T. J. Green and V. Tannen. Annotated XML: queries and provenance. In *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2008, June 9-11, 2008, Vancouver, BC, Canada*, pages 271-280, 2008.

[11] T. J. Green, G. Karvounarakis and V. Tannen. Provenance semirings. In *PODS*, pages 31-40, ACM, 2007.

[12] A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270-294, 2001.

[13] G. Joshi-Tope, M. Gillespie, I. Vastrik, P. Eustachio, E. Schmidt, and et al. Reactome: A knowledge base of biological pathways. *Nucleic Acids Res*, 33 (DATABASE ISS.), 2005.

[14] D. Longo and J. Drazen. Data Sharing. *N Engl J Med [Internet]*, 374(3):276-277, 2016.

[15] A. Rajaraman, Y. Sagiv and J. D. Ullman. Answering Queries Using Templates with Binding Patterns. In *Proc. of the 14th Symposium on Principles of Database Systems*, pages 105-112, 1995.

[16] C. Southan, J. L. Sharman, H. E. Benson, E. Faccenda, A. J. Pawson, S. P. H. Alexander, O. P. Buneman, A. P. Davenport, J. C. McGrath, J. A. Peters, M. Spedding, W. A. Catterall, D. Fabbro, J. A. Davies and Nc-Iuphar. The IUPHAR/BPS Guide to PHARMACOLOGY in 2016: towards curated quantitative interactions between 1300 protein targets and 6000 ligands. *Nucleic Acids Research*, 44:1054-1068, 2016.

[17] C. Torniai, D. Bourges-Waldegg and S. Hoffmann. eagle-i: Biomedical research resource datasets. *Semant Web*, 6:139-146, 2015.