Automating Data Citation in CiteDB

A. Alawini, S. B. Davidson, W. Hu, Y. Wu
Dept. Computer and Information Science
University of Pennsylvania
{alawini, susan, huwei, wuyinjun}@seas.upenn.edu

ABSTRACT

An increasing amount of information is being collected in structured, evolving, curated databases, driving the question of how information extracted from such datasets via queries should be cited. While several databases say how data should be cited for web-page views of the database, they leave it to users to manually construct the citations. Furthermore, they do not say how data extracted by queries other than web-page views – *general queries* – should be cited. This demo shows how citations can be specified for a small set of views of the database, and used to *automatically generate citations for general queries* against the database.

1. INTRODUCTION

Data citation is of growing concern for owners of curated, on-line scientific databases. Since much of the content is contributed by members of the community and curated by experts, there is growing recognition that the data extracted by a query should recognize the efforts of the individuals (authors) involved in creating the dataset. Owners may therefore want a citation to data extracted from their database by a query to include snippets of information about the authors, identifying features of the data extracted, the query used to extract the data, the database version, and so on.

Currently, several databases describe (in English) what citations should look like for data displayed as web page views of the database (e.g. the Reactome Pathway database [1] and eagle-i [2]), but few databases generate the citation automatically. A notable exception to this is the IUPHAR/BPS Guide to Pharmacology [3] (GtoPdb), in which the citations for web-page views of the database are hard-coded in the web page results. However, none provide a citation for *general queries* over the database, i.e. those which do not correspond to web page views of the database.

This problem is especially hard because, unlike traditional publications which have a fixed granularity to which citations can be attached (e.g. a paper in a conference proceedings, or chapter in a book), the granularity of data varies when retrieved by a query over a database. Since there are a potentially infinite number of queries, each accessing and generating different subsets of the database, it is impossible to explicitly attach a citation to every possible result set and/or query.

CiteDB addresses the problem of generating citations to general queries over a relational database by allowing owners to specify citations for views of the database which represent frequent queries (e.g. web page views), and uses these *citation views* to automatically construct citations for data returned by general queries. It does so by determining which view tuples contributed to the result, and combining their associated citations to form a citation for the query result. The manner in which citations are combined are *policies* specified by the owner.

Demonstration. We will demonstrate CiteDB on GtoPdb, a relational database that contains expertly curated information about drugs in clinical use and some experimental drugs, together with information on the cellular targets of the drugs and their mechanisms of action in the body. Currently, GtoPdb generates citations to a subset of the possible queries against the underlying relational database, i.e. those corresponding to web-page views of the data. We will show how these citations can be 1) separately specified by the database administrator (DBA) rather than being embedded in web-page code, and 2) used to automatically construct citations to general queries against the database. Furthermore, we will show how users can browse the citations attached to tuples or subsets of tuples within a query result.

Participants will be invited to construct general queries against GtoPdb through a QBE-like interface, and view query results along with the citation. They may also specify additional citation views for GtoPdb.

Related work. Core principles [7, 8] and standards [5] have been proposed for data citation. A prototype citation system for relational databases is described in [9], which returns a stable identifier along with the query result and ensures recoverability of data as of the query time.

The idea of a *rule-based citation system* to provide citations that include snippets of information for hierarchically structured data sources was first proposed in [4]. This was

[1] http://www.reactome.org/pages/documentation/citing-reactome-publications/
[2] https://www.eagle-i.net/get-involved/for-researchers/citing-an-eagle-i-resource/
[3] http://www.guidetopharmacology.org/

Proceedings of the VLDB Endowment, Vol. 10, No. 12

extended in [3] with a notion of *views* defining citable units, and applied to the eagle-i resource discovery dataset in [2]. The notion of citations as *fine-grained annotations* was introduced in [6], and forms the basis for this demo.

2. TECHNICAL BACKGROUND

Throughout this discussion, we will use GtoPdb as the example. In GtoPdb 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 into sub-families and so on down to individual drug targets and drugs. The content of a particular family "landing" page is curated by a committee of experts; a family may also have a "detailed introduction page" which is written by a set of contributors, who are not necessarily the same as the committee of experts for the family.

The citations for these views of the database, which are parameterized by the family id, therefore vary: the citation for GtoPdb as a whole is a traditional paper written by the database owners, a citation to a family page includes the committee members who curated the content, and a citation to a family detailed introduction page includes the contributors who wrote the content.

The (simplified) GtoPdb schema we will use is, where keys are underlined:

Family(FID, FName, Type)
FamilyIntro(FID, Text)
Person(PID, PName, Affiliation)
FC(FID, PID), FID references Family,
PID references Person
FIC (FID, PID), FID references FamilyIntro,
PID references Person

We now describe our model and implementation.

2.1 Model

We start with a set of *view definitions* and *citation queries*, that are specified by the database owner. Each citation query is associated with a *citation function*, which takes the output of the citation query as input, and outputs a citation in some appropriate format (e.g. human readable, BibTex, RIS or XML).

Internally, the view definition and citation query are Conjunctive Queries (see [1] for an overview) that are optionally *parameterized* by one or more variables. A parameterized view creates a set of views, one for each possible choice of parameters. The number of such views is therefore instance-dependent.

For example, the following (parameterized) view creates a separate view instance for each tuple in `Family`:

λ `FID. V1(FID,FName,Type):- Family(FID,FNname,Type)`

Each view is associated with one or more citation queries. To ensure that the citation is the same across all tuples in the view, the parameters of the citation query must be a subset of the distinguished variables of the view definition.

For example, the following citation query retrieves the names of all committee members for a given `FID` and can be associated with `V1`,:

λ `FID. C1(PN):- FC(FID,PID), Person(PID,PN,A)`

We could also associate a citation query with *no* parameters to `V1`, for example, a citation for the traditional reference paper for GtoPdb as a whole.

To give a semantics for citations for *general queries*, we use the following intuition: *If a view tuple can be used to create a tuple in the query result, then the result tuple carries the view tuple's citation annotation.* We formalize this next.[4]

We assume that all queries (including view definitions) use fresh variables in every position; any *local* constraints on variables (i.e. those involving a single variable) and *global* constraints (i.e. those involving more than one variable) are expressed as non-relational subgoals of a query.

DEFINITION 2.1. **Query Extension** *Given a query*

$$\mathtt{Q}(\bar{\mathtt{X}}) : -\mathtt{B}_1(\bar{\mathtt{X}_1}), \mathtt{B}_2(\bar{\mathtt{X}_2}), \ldots, \mathtt{B}_\mathtt{m}(\bar{\mathtt{X}_\mathtt{m}}), \mathtt{condition}(\mathtt{Q})$$

where condition(Q) are the non-relational subgoals, let $\bar{X}' = \cup_{i=1}^{m} \bar{X}_i$. *Then the* extension of Q, Q_{ext}, *is*

$$\mathtt{Q}_\mathtt{ext}(\bar{\mathtt{X}}') : -\mathtt{B}_1(\bar{\mathtt{X}_1}), \mathtt{B}_2(\bar{\mathtt{X}_2}), \ldots, \mathtt{B}_\mathtt{m}(\bar{\mathtt{X}_\mathtt{m}}), \mathtt{condition}(\mathtt{Q})$$

Since a view definition is also a query, we use the same notion for V_{ext}.

DEFINITION 2.2. **View Mapping** *Given a view definition* `V` *and query* `Q`

$$\mathtt{V}(\bar{\mathtt{Y}}) : -\mathtt{A}_1(\bar{\mathtt{Y}_1}), \mathtt{A}_2(\bar{\mathtt{Y}_2}), \ldots, \mathtt{A}_\mathtt{k}(\bar{\mathtt{Y}_\mathtt{k}}), \mathtt{condition}(\mathtt{V})$$
$$\mathtt{Q}(\bar{\mathtt{X}}) : -\mathtt{B}_1(\bar{\mathtt{X}_1}), \mathtt{B}_2(\bar{\mathtt{X}_2}), \ldots, \mathtt{B}_\mathtt{m}(\bar{\mathtt{X}_\mathtt{m}}), \mathtt{condition}(\mathtt{Q})$$

a **view mapping** *M from V to Q is a tuple* (h, ϕ) *in which:*

- *h is a homomorphism which maps each relational subgoal* A_i *in V to a relational subgoal* B_j *in Q with 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 subgoal B_j *of Q is* covered *iff* $h(A_i) = B_j$ *for some i.*

A view may be in zero or more view mappings for a given query. Since each view is associated with a citation (denoted $Cite(V)$), we can use this to cite subgoals of Q that are covered by M (denoted $Cite(M)$).

DEFINITION 2.3. **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 either a distinguished variable or appears in condition(Q).*

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 $t \in Q_{ext}(D)$ according to Definitions 2.2 and 2.3. We then combine different view mappings from $\mathcal{M}(t)$ to create a *covering set* of views for t.

DEFINITION 2.4. **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*

[4]This semantics is slightly different from that in [6].

Table 1: Sample table for base relation Family

Family_id	name	Type	view vector
58	n1	gpcr	V1,V4('gpcr')
59	n2	gpcr	V1, V4('gpcr')
60	n3	lgic	V1,V4('lgic')
61	n4	vgic	V1,V2,V4('vgic')
62	n5	vgic	V1,V2, V4('vgic')

Table 2: Sample table for base relation FamilyIntro

Family_id	Text	view vector
58	tx1	V3(58),V4(␣)
60	tx2	V3(60),V4(␣)
61	tx3	V3(61),V4(␣)
62	tx4	V3(62),V4(␣)

- *No $V \in \mathcal{M}(t) - C$ can be added to C to cover more subgoals of Q; and*
- *No $V \in C$ can be removed from C and cover the same subgoals of Q.*

Note that for each tuple t there may be a *set* of covering sets, $\{C_1, ..., C_k\}$. In each $C_i = \{M_1, M_2, \ldots, M_l\}$, the citation views are *jointly* used (denoted *) to construct a citation for t: the citation $Cite(C_i)$ for t is $Cite(M_1) * \cdots * Cite(M_l)$. The citations from each C_i are then *alternately* used (denoted $+^R$) to construct a citation for t: $Cite(t) = Cite(C_1) +^R \cdots +^R Cite(C_p)$.

The result of Q is obtained by projecting Q_{ext} over Q's distinguished variables: $Q(D) = \Pi_S Q_{ext}(D)$. Thus a tuple $t \in Q(D)$ may be derived from multiple tuples in $Q_{ext}(D)$. The annotations from all derivations of t are therefore combined to form a citation for t using the abstract operator $+$, indicating *alternate derivations*. To create the citation for the query result, the annotations of all tuples in the result are combined using the abstract operator *Agg*. The abstract operators *, $+^R$, $+$ and *Agg* are *policies* to be specified by the database owner, and could be union, the "best" in some ordering over view mappings, or some form of join.

2.2 Implementation

The implementation starts by expanding the schema of each relation with a `view vector` column, which identifies all views in which a tuple *potentially* participates. Thus when a view $V : -B_V$ is added to the database schema, V is added to the view vector of each tuple t occurring in each relation $R \in B_V$ such that t satisfies *local predicates* for R. Any *global predicates* comparing variables from different relations (e.g. joins) will be checked at query time.

For example, suppose we had the following views:

```
V1(FName)          :- Family(FID, FName, Type)
V2(Type)           :- Family(FID, FName, Type), FID>60
λFID.V3(FID,Text)  :- FamilyIntro(FID, Text)
λType.V4(Type)     :- Family(FID1,FName,Type),
                      FamilyIntro(FID2,Text), FID1 = FID2
```

Expanded sample instances for `Family` and `FamilyIntro` are shown in Tables 1-2. Note that for each parameterized view, we add all parameter values local to the tuple to the view id. View parameters that are not local to the tuple are left unspecified. For example, when annotating tuples in `FamilyIntro` with $V4$, a placeholder is added for the parameter `Type` since it comes from `Family`.

Table 3: Final query result

Ty	Cite(t)
gpcr	Cite(V4('gpcr'))
lgic	Cite(V4('lgic'))
vgic	(Cite(V2)* Cite(V3(61)) $+^R$ Cite(V4('vgic'))) + (Cite(V2)* Cite(V3(62)) $+^R$ Cite(V4('vgic')))

When a query $Q : -B_Q$ is submitted, we first remove views with no view mappings. The query Q is then extended to include 1) all head variables of Q and the lambda variables under all the possible view mappings; 2) the view vectors of every base relation occurring in B_Q; and 3) columns representing the truth value of every local and global predicate under every possible view mappings (details omitted). The extended query, called Q_{ext1}, is then executed over the database instance D as an SQL query which calculates the truth value of local and global predicates, yielding an instance $Q_{ext1}(D)$ over which the first phase of citation reasoning occurs.

In this phase, the valid view mappings for each view vector for each tuple $t \in Q_{ext1}(D)$ are first calculated. A multi-relation view mapping is valid iff all *global predicates* under this mapping evaluate to true. We will remove view vectors without any valid view mappings. Covering sets of view mappings for t are then calculated by picking one view mapping from each remaining view vector and combining them using *. Different covering sets are combined using $+^R$. The output of this phase is a relation $Q_{ext2}(D)$ with an additional column containing $Cite(t)$.

To improve performance, $Q_{ext1}(D)$ can be grouped into *sets* of tuples satisfying the same local and global predicates. Reasoning can then be performed for one tuple in the group, and the result applied to other tuples in the group. This results in considerable performance gains.

The next phase projects $Q_{ext2}(D)$ over the distinguished variables of Q, and calculates the $+$ (alternate use) derivations of valid citations: For each $\{t1, ..., tk\} \subseteq Q_{ext2}(D)$ that contain the same values for distinguished variables in Q, construct the $+$ of their valid citations. The result of this phase will be the query result of Q, augmented with an additional column containing the valid citations for each tuple.

For example, consider the following query:

```
Q(Ty):- Family(F1,N,Ty), FamilyIntro(F2,Tx), F1 = F2
```

There is no valid view mapping for $V1$ since the second condition in Definition 2.3 cannot be satisfied. Any view whose body contains a relation other than `Family` or `FamilyIntro` would also be removed (there are none in our example).

The result of executing phases 1 and 2 over Q is shown in Table 3. The citation for the query result is constructed using the *Agg* of citations for each tuple in the result. For each $Cite(t)$ in $Q(D)$ as well as the aggregation results, we extract snippets of information using the associated citation views and apply the specified policies for *, $+$, $+^R$ and *Agg* to generate the final citation.

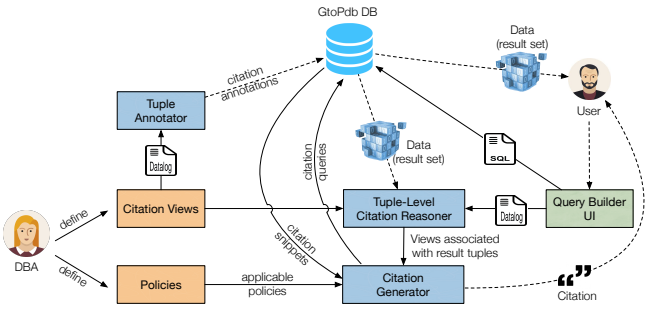

Figure 1: CiteDB Framework

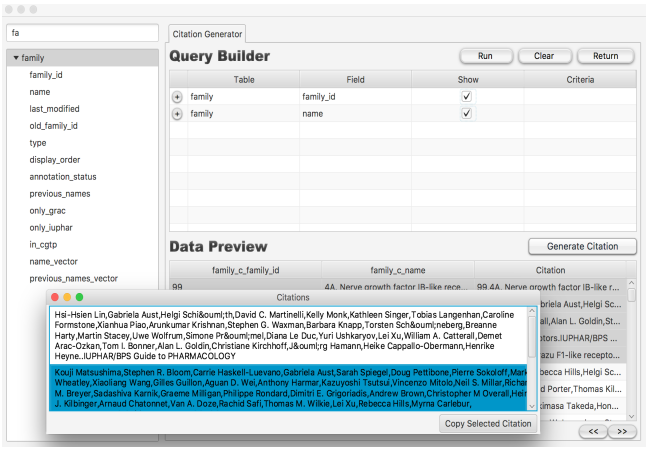

Figure 2: Query Builder

3. SYSTEM OVERVIEW

CiteDB is implemented in Java 8, and its backend storage is embedded in the GtoPdb database (PostgresSQL). An overview of CiteDB is shown in Figure 1. The DBA first specifies citation views and policies for how they are to be used in constructing citations for general queries such as joint- and alternates-use policies. Next, the *tuple annotator* annotates the tuples of each base relation with the views in which they may appear. These annotations are used to construct a citation for the result set of a general query.

When a user submits a query Q through the *query builder UI*, a QBE-like graphical user interface, CiteDB translates Q into Datalog. The query is analyzed by the *citation reasoner*, which translates Q_{ext1} into SQL for evaluation, constructs and evaluates Q_{ext2} based on the result of Q_{ext1}, and constructs the valid view combinations for each result tuple. Citation queries for the chosen views are then executed in the GtoPdb database to retrieve the appropriate snippets of information, and the information combined using the specified *, + and $+^R$ policies. The citation is then returned to the user along with the data.

Query Builder UI. CiteDB has two types of query builder interfaces, which are implemented using the Java SWING library. First, a *user interface*, which enables a user to 1) construct and execute a general query, and 2) generate a citation for its result set. Second, a *DBA interface,* which allows the database owner to define frequent database views along with their corresponding citation views.

4. DEMONSTRATION

For the demo, we connected CiteDB to the GtoPdb database to extend its functionality by allowing users to submit general queries and have them cited. Attendees will be invited to use CiteDB to construct queries and submit them to GtoPdb, and view citations for the returned results via CiteDB's UI. We will also demonstrate how DBAs can use the administrator console to create view and citation queries.

We will first illustate the use of CiteDB and its UI by showing users how to create queries and generate citations for them. Next, we will invite the audience to try CiteDB's query builder interface by creating new queries. Then, users can examine the results of their queries and click "Generate Citations" to automatically construct the result citation (as shown in Figure 2).

To highlight the capabilities of the tuple-level citation approach in CiteDB, we will show users how valid view combinations are associated with a single row, a subset of the rows or the whole result set. This made possible by the tuple-level annotations that are used to determine how to cite a query results at various levels of granularity.

Finally, to help our audience better understand the tuple-level citation approach of CiteDB, we will demonstrate how database owners can manage view and citation queries via the administrator console of CiteDB.

Acknowledgments. This work has been partially funded by NSF IIS 1302212, NSF ACI 1547360, and NIH 3-U01-EB-020954-02S1.

5. REFERENCES

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

[2] A. Alawini, L. Chen, S. B. Davidson, N. Portilho, and G. Silvello. Automating data citation: the eagle-i experience. In *JCDL*, 2017.

[3] P. Buneman, S. B. Davidson, and J. Frew. Why data citation is a computational problem. *Communications of the ACM (CACM)*, 59(9):50–57, 2016.

[4] P. Buneman and G. Silvello. A Rule-Based Citation System for Structured and Evolving Datasets. *IEEE Data Eng. Bull.*, 33(3):33–41, 2010.

[5] DataCite. DataCite metadata schema for the publication and citation of research data. `http://schema.datacite.org/meta/kernel-3/doc/DataCite-MetadataKernel_v3.1.pdf` (accessed Nov 2016), October 2014.

[6] S. B. Davidson, D. Deutsch, T. Milo, and G. Silvello. A model for fine-grained data citation. In *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Online Proceedings*, 2017.

[7] FORCE-11. *Data Citation Synthesis Group: Joint Declaration of Data Citation Principles*. FORCE11, San Diego, CA, USA, 2014.

[8] C.-I. T. G. on Data Citation Standards and Practices. *Out of Cite, Out of Mind: The Current State of Practice, Policy, and Technology for the Citation of Data*, volume 12. September 2013.

[9] S. Pröll and A. Rauber. Scalable data citation in dynamic, large databases: Model and reference implementation. In *Proc. of the 2013 IEEE International Conference on Big Data*, pages 307–312, 2013.