A Framework for Fine-grained Data Integration and Curation, with Provenance, in a Dataspace

David W. Archer, Lois M. L. Delcambre, David Maier

Department of Computer Science, Portland State University
Portland, OR 97207 USA

{darcher, lmd, maier @cs.pdx.edu}

Abstract

Some tasks in a dataspace (a loose collection of heterogeneous data sources) require integration of fine-grained data from diverse sources. This work is often done by end users knowledgeable about the domain, who copy-and-paste data into a spreadsheet or other existing application. Inspired by this kind of work, in this paper we define a *data curation setting* characterized by data that are explicitly selected, copied, and then pasted into a target dataset where they can be confirmed or replaced. Rows and columns in the target may also be combined, for example, when redundant. Each of these actions is an integration decision, often of high quality, that when taken together comprise the provenance of a data value in the target. In this paper, we define a conceptual model for data and provenance for these user actions, and we show how questions about data provenance can be answered. We note that our model can be used in automated data curation as well as in a setting with the manual activity we emphasize in our examples.

1. Introduction

When a user is confronted with a potentially large number of diverse data sources of variable maturity or quality, i.e., a dataspace [1], she may have limited tools available to integrate or query data from these sources. A user typically uses a weak tool such as a spreadsheet to gather data and almost always uses a copy-and-paste action to transfer data from sources into an integrated target dataset. This kind of work differs substantially from traditional information integration, including update exchange, as discussed in the literature [2,3,4,5], though aspects of it have been addressed in literature concerning curated databases [10,11,18,19]. Perhaps the most compelling differences between this type of work and traditional integration are that the user works with individual data values, rather than sets of rows at a time, and that provenance is needed at the level of these individual values. These differences and others we discuss in Section 2 lead us to describe and address a *data curation setting*.

As an example, consider Anne, a battalion information officer working in a theatre of military operation. Anne gathers and organizes information from a variety of sources to help her commanders make decisions. A typical task for Anne might be to assemble a table of casualty information due to explosive device incidents during the prior week in a given patrol area. Data sources for assembling this report may include incident databases from friendly forces in the area, e-mail exchanges with local police, patrol logs for the week, and medical-team records. Anne uses her personal knowledge of the operations area, recent events, reliability of sources, and her commander's preferences for data as she works. She may begin gathering data by writing a query against an existing database, as in traditional information integration. However, she then proceeds along non-traditional lines: adding rows to her table as she identifies incidents of interest; selecting data values from external sources and inserting them into rows and columns in the evolving data table by using copy-and-paste actions; adding columns to her table as needed; manually overriding data values directly or with data from other sources; merging rows that she realizes represent the same incident into new rows (entity resolution), and choosing correct values for each column in the resulting row from those being merged; combining information from multiple columns that she realizes are redundant (attribute resolution), and choosing correct values for each row in the resulting column from those being merged; confirming data elements she knows to be accurate; correcting mistakes; and issuing further updates.

Each of Anne's actions is an information integration decision, and each data value in her report is the product of such decisions. This provenance is often lost with current tools, but is key to answering questions such as, *"Where did we get the date for incident 105?"*

We seek to enable this kind of data integration and provenance exploration with minimal interference with a user's workflow. We make the following contributions toward our goal in this paper. We elaborate the example above in order to describe the differences between this and traditional information integration settings. We extend the set of user integration actions defined in our prior work [6,7] to include user confirmation, so that a user can express confidence in a data value, even if the value is not changing. We show how to derive from a record of user integration actions the provenance, plurality of support, and trust of each data value in the assembled data set, and how to use this information to answer questions that might arise in our setting.

Although our work was inspired by manual curation, it is applicable to an automated or mixed manual-plus-automatic setting. A user can copy data in bulk from a query answer or view into the data curation setting. Whether manual or automated, we only require that the "paste" part of the action specify precisely which cell is the target for each value. In this paper, we describe the actions in a data curation setting as if a user performed them manually. The remainder of our paper is organized as follows. In Section 2, we elaborate our example of a data curation setting and describe a conceptual model to support this setting, including the new user confirmation action. In Section 3, we describe an implementation approach to support our conceptual model. In Section 4, we relate copy-and-paste, entity resolution, and attribute resolution operations to data provenance. In Section 5, we show how our model is able to answer interesting questions about provenance. In Section 6, we evaluate our approach with regard to storage overhead. Section 7 discusses related work and Section 8 concludes.

2. Use Model for a Data Curation Setting

We assume that Anne uses a tool for data integration that records each of her actions. In our example (and in the prototype implementation of our data model), Anne's actions may include: initially populating a data table via a query over an external source; adding or deleting rows (entities) and columns (attributes); copying data values from external information sources and pasting them into rows and columns of the table (either updating or adding information); and performing entity and attribute resolution operations. We have not included here the notion of copying a value from within a data table and pasting it elsewhere in the table, as Buneman et al. do [10,11]. Suppose Anne writes a query to gather some initial data, with result as follows:

IncidID	Mapl1	Mapl2	T	Dev
101	34.3998	70.4986		ied
102			2	art
103	34.3998	70.4985	1	

Here, IncidID is Anne's identifier for incidents, Mapl1 and Mapl2 are map coordinates, T is some unspecified attribute, and Dev is the type of munition involved. Implicit in this table (and not visible to the user), is a unique, system-generated key for each row in the table, used by the system for recording the history of data manipulation in the row. Next, Anne gathers more data using copy-and-paste, and finds information on casualties and dates for the incidents (adding new columns Cas and Date), along with other data (new columns ME, MLat, and MLng), which she does not fully understand yet. Her data table now is shown in Table 1.

At this point, Anne determines that incidents 101 and 103 are the same, and that incidents 102 and 104 are the same. Selecting "entity resolution" from a menu, she creates a new entity from incidents 101 and 103, and as part of the operation she selects, on an attribute-by-attribute basis, which values to propagate from the original incidents into the new (merged) row whenever the source rows disagree. The original rows are then hidden from Anne's view, leaving only the new entity. She then resolves incidents 102 and 104. Choices made during the resolutions are highlighted in Table 2.

ID	Mapl1	Mapl2	T	Dev	Cas	ME	MLat	MLng	Date
101	34.3998	70.4986	1	ied	3	3	3423.99	7029.92	03mar
102	34.3996	70.4985	2	art	7	6	3423.96	7029.90	05mar
103	34.3998	70.4985	1	ied	4	3	3423.99	7029.91	03mar
104	34.3996	70.4985	2	msl	7	6	3423.96	7029.90	04mar
105	34.3994	70.4988	1	ied	12	12	3423.94	7029.88	06mar

Table 1. Intermediate data table after first round of copy-and-paste integration.

ID	Mapl1	Mapl2	T	Dev	Cas	ME	Mlat	MLng	Date
101	34.3998	70.4986	1	ied	**4**	3	3423.99	7029.92	03 mar
104	34.3996	70.4985	2	**art**	7	6	3423.96	7029.90	**04 mar**
105	34.3994	70.4988	1	ied	12	12	3423.94	7029.88	06 mar

Table 2. Data table after initial entity resolution.

Next, Anne finds that columns Mapl1 and Mlat are the same geographical location data, though in different formats. Similarly, Mapl2 and MLng are redundant. She then determines that Cas (Casualties) and ME (which she discovers to mean the count of soldiers MedEvac'd from the scene) also represent the same information, which she chooses to call Injured. Resolving these columns proceeds in fashion similar to the entity resolution operations. Anne chooses new, meaningful column names for the new columns, and her table takes the form:

ID	Lat	Long	T	Dev	Injured	Date
101	**34.3998**	70.4986	1	ied	**4**	03mar
104	**34.3996**	70.4985	2	art	**6**	04mar
105	**34.3994**	70.4988	1	ied	12	06mar

Review of the data against other information reveals inconsistencies. She changes the "Injured" column of incident 105 to 9 from 12, and the date of incident 105 to 07mar. From personal knowledge, she confirms the "injured" column of incident 104 as 6. At last, her report is in final form:

ID	Lat	Long	T	Dev	Injured	Date
101	34.3998	70.4986	1	ied	4	03mar
104	34.3996	70.4985	2	art	**6**	04mar
105	34.3994	70.4988	1	ied	**9**	**07mar**

From the point at which Anne began gathering data to the point at which the report is in final form, 95 recordable history actions occurred. At this point, Anne might need to examine the history behind selected data. She might for example ask the following questions:

- *"Which incidents were combined from others in the table?"*
- *"Which incidents in the table had casualty data that was inconsistent and was corrected?"*
- *"For each incident merged from redundant incidents, what were the incident IDs of the original rows that we chose to resolve?"*
- *"How many information sources have we found to support the casualty count for the IED incident in row 112?"*
- *"What other names did column Long in this table have in earlier versions of the data you gathered?"*
- *"What information about incident 109, if any, was derived differently than the rest?"*

To answer these, Anne can click on a data value in her table and be presented with a graph of the provenance for the value, as shown in Figure 1. This graph shows not only the origins (external sources) for the selected data value, but also the intermediate values that contributed to its current state. All user actions that led to the current state are also shown, including which values were preferred over others during resolution operations, as well as confirmations made by users. For example, the figure shows that the value for incident 104 in the Injured column was most recently derived by attribute resolution (A), where the value at {104, wounded} in the table was preferred by the user over the value at {104, cas} (the dashed edge indicates non-preferred). The figure also shows that the current value was confirmed by user input (X). The conceptual model visible to the user in our setting consists of the data table she constructed, along with a provenance graph for each "cell" in the table.

This setting has a number of novel characteristics:

- Access to the collection of data sources found in a dataspace may be varied. For example, sources may come and go over time. In addition, we may wish to produce a snapshot in time of a dataspace that is dynamic. Because of this requirement, queries are answered using only the target instance.
- Information is most often integrated at the data level by selecting individual data values from sources. However, traditional integration may also play a role.

- A data value (once selected) is inserted into a specific location in the target instance, identified by an internal, unique row-id and a column name (distinguished by means of the user pointing to a table cell). The user is implicitly joining each new value with a row that already exists and specifying a particular existing column into which the data value goes.

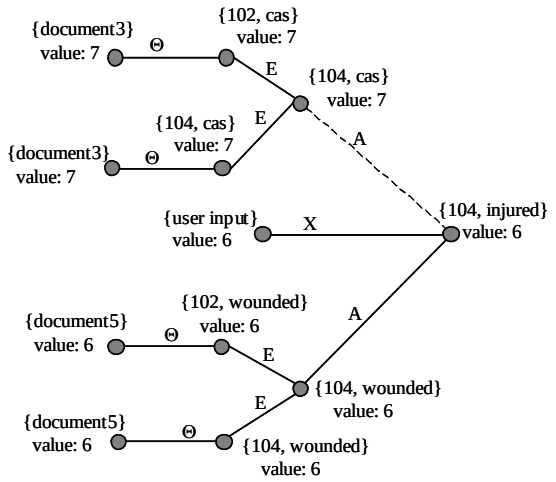

Fig. 1. Provenance tree for (104,"Injured") where the current value is 6. Edges represent user actions (A for attribute resolution, E for entity resolution, X for confirmation, Θ for update). Nodes show state of values in the data table. Non-preferred edges are shown using a dashed line.

- The target instance may include new data inserted from the user's knowledge and not from an identified data source.
- The user can confirm that a data value in the target is valid, either by pasting in the same value from a second source, or by locally confirming the value.
- A data value gathered by a user may arise from multiple sources, be the result of multiple human judgements, have multiple attribute name designations, and have multiple entity associations during the integration process. As a result, a user may want to review the reliability of her data, understand how much support there is for a given data value, and possibly review the set of values that a given data value has had over time.

3. An Implementation Approach

We developed a prototype end-user application called CHIME (Capturing Human Intension Metadata with Entities) that supports the kind of integration task outlined above. CHIME supports creation and manipulation of a single, entity-centric data table, while capturing the history of manipulation operations (insertions and updates, entity resolutions, attribute resolutions, and confirmations) that comprise the provenance of data in the table. Rows in this table correspond to entity instances, while columns correspond to attributes of the entities. (In this paper, we use the term "entity" to refer to an entity instance.) Each entity in a CHIME dataset (i.e., the data shown in one row) has an internally generated, unique identifier, called KeyVal in the schema, which is not made visible to the user.

CHIME data can be modelled by two relations. We define schema R with instance r to represent entities:

- KeyVal, a unique system-generated identifier that functions as a candidate key for R
- {$Attr_1$, Col.Visible1, $Attr_2$, Col.Visible2…$Attr_N$, Col.VisibleN}, where each Attr is a user-defined attribute name and each Col.Visible a Boolean that specifies whether Attr is visible to the user and eligible for CHIME operations.
- Row.Visible, a Boolean attribute that specifies whether this tuple in r is visible to the user and eligible for CHIME operations.

We define a history table schema H, with instance h. Each user action on the data relation r adds new entries to h. H has attributes:

- SeqNum, a monotonic integer (e.g., a timestamp)
- KeyVal, a foreign key referencing R
- AttrName, an attribute with domain {$Attr_1$, $Attr_2$, …$Attr_N$} from R
- AttrVal, the value given to attribute AttrName in the row in r with key KeyVal as a result of this action
- Action, an attribute with domain {A, E, Θ, X} (for attribute resolution, entity resolution, update, and user confirmation, respectively)
- Preferred, an attribute with domain {Left, Right, Both} that indicates which parent tuple or column was preferred for item (KeyVal, AttrName) by the user during a resolution operation
- LeftParent, an attribute with domain {dom(AttrName) ∪ dom(KeyVal) ∪ NULL}, indicating one parent row (for entity resolution) or column (for attribute resolution), or NULL otherwise

- RightParent, an attribute with domain {dom(AttrName) ∪ dom(KeyVal) ∪ NULL }, indicating the other of two parents resolved to form the new data value
- Source, a string-valued attribute with domain consisting of the distinguished value "User-supplied" and names of data sources

Taken together, Seqnum, KeyVal, and AttrName form a candidate key for H. Note that multiple entries in H may have the same SeqNum. For example, all entries associated with a resolution operation have the same timestamp, though each entry affects a different target data value.

We envision a user interface (much like our CHIME prototype interface) where data in r is shown in table form, with only rows and columns with Visible flags set to True visible to the user. In the interface, the user can easily see the corresponding data from h in the form of a provenance graph for any data value shown, e.g. by clicking on the data value.

4. Relating User Actions to Provenance

We map information from CHIME actions to fields of entries in the CHIME history table (h) in order to compute provenance and answer user questions. Copy-and-paste actions that insert new (or different) values into a data value are mapped to entries in h as follows:

- SeqNum = (automatically generated) sequence number of this update in the overall history
- KeyVal = internal identifier of target row
- AttrName = name of target column
- AttrVal = value pasted
- Action = Update
- Preferred = NULL
- LeftParent = NULL
- RightParent = NULL
- Source = source identifier or "User-supplied"

Users can copy-and-paste the same value into a data value again, possibly from a different source. In this case, we add an entry to the history table, recording the user action as Confirm. The user may also Confirm a value directly without a copy-and-paste operation, when there is no external source.

Entity resolution in our model consists of merging two entities into a new entity, then making the original entities not visible. For each attribute (column) of the resulting entity where source data values differ (note that the schema of source entities and result entity are the same), the user must choose which source attribute value to propagate. When source attribute values are identical, we consider both to be propagated, for provenance purposes. Attribute resolution in our model works in a similar fashion. Two attributes are merged into a new attribute, and the two original columns are made non-visible. In rows where the two source values differ, the user chooses column values to propagate to the new column. We map entity resolution and attribute resolution operations to entries in h as follows:

- SeqNum = (automatically generated) sequence number of this update in the overall history. (Because resolutions affect multiple data values, and it is useful to identify all effects of a resolution, all such effect are recorded with one SeqNum)
- KeyVal = internal identifier for row of the data value resulting from entity or attribute resolution
- AttrName = column name of the data value resulting from entity or attribute resolution
- AttrVal = value of the data value after resolution
- Action = Entity or Attribute resolution
- Preferred = Left, Right, or Both referring to one or both of the Parent values below
- LeftParent = one of the KeyVals (if entity resolution) or column names (if attribute resolution) involved in entity or attribute resolution
- RightParent = the other KeyVal or column name involved in entity or attribute resolution
- Source = NULL

5. Computing and Expressing Provenance

In this section we explore the use of the history relation to generate provenance graphs. We then define a set of predicates and algorithms to traverse these graphs in order to answer user questions.

We define a *provenance tree* $T_P(V,E)$ for a data value in the user's table as a directed acyclic graph with V a set of vertices and E a set of edges. A vertex $v \in V$ in T_p corresponds to

- the current state of the data value of interest, if the vertex is the root of T_p
- a prior state (ancestor) of the data value of interest, if the vertex is neither the root nor a leaf in the tree
- the source of an update or confirmation, if the vertex is a leaf in the tree (either the name of an external source, or a constant user-supplied value when the user confirms the value directly)

Each edge $e \in E$ exits a vertex in V (other than the root) and enters a distinct, non-leaf vertex in V.

A provenance graph represents an Update operation by adding a leaf vertex to represent the source from which the data was copied (or to represent a constant supplied directly by the user), and an Update edge from the source vertex to the affected data value vertex. If the data value is new (that is, if the Update corresponds to insertion of a data value for the first time), a vertex is added to represent the data value in the table. A Confirm operation is represented similarly, but the connecting edge is labelled Confirm. If two data values are merged during an entity or attribute resolution, a new vertex is added to represent the resulting merged data value. E includes two edges for the operations, one exiting each vertex representing a "parent" data value and both entering the vertex representing the new (merged) data value. Where the two values resolved are not equal, each edge indicates whether the parent vertex held the value preferred by the user, or the overridden (not preferred) value. In a resolution with identical parent data values, both edges are preferred. In the absence of resolution operations, a provenance graph for a data value consists only of a root vertex and any leaf (source) vertices. Intermediate (non-root, non-leaf) vertices are only introduced by resolutions.

We represent $T_p(V,E)$ for one data value as a pair of relations, Node and Edge, that are temporary, and generated on-the-fly when the provenance graph is created by user request. Node includes attributes:

- **Nodenum,** a candidate key for Node
- **KeyValue,** an foreign key referencing KeyVal in H
- **AttrName**, an attribute with domain {$Attr_1$, $Attr_2$, … $Attr_N$} from R, equal to AttrName in the corresponding entry in h
- **AttrVal**, = AttrVal in the corresponding entry in h
- **Source**, = Source in the corresponding row in h

The attributes for Edge are:

- **Descendant** – a foreign key referencing Node
- **Ancestor** –a foreign key referencing Node
- **Action**, with domain {Attribute Resolution, Entity Resolution, Update, Confirm} equal to Action in the entry in h for the Descendant's KeyValue
- **Preferred**, a Boolean value equal to TRUE if Ancestor's value was preferred during the action creating Descendant, and FALSE otherwise. This attribute is ignored if Action is not a resolution action

We construct $T_p(V,E)$ for a data value in r (specified by the user selecting a row and column) by retrieving from the history table the list of actions performed on the value and its ancestors. Figure 2 shows an abstraction of the Prolog implementation of our algorithm. The *get_history* predicate retrieves from h a list of user actions performed on the value at row-id *Key* and column *Attr*. The *setpref* predicate sets the value of the "Preferred" indicator for edges in T_p. Our algorithm constructs a textual version of T_p via *write* statements.

```
make_ptree(Key, Attr, Root) :-
        get_history(Key, Attr, [H|Rest]),
        history(H,_,_,Val,_,_,_,Src),
        build_ptree(Key, Attr, H, [H|Rest], true),
        write("node(",H,Key,Attr,Val,Src,").").
        Root is H.
% most recent action was an update
build_ptree(Key, Attr, Root, [H|Rest], Pref_flag) :-
        history(H,_,_,Val,u,_,_,_,Src),
        S is 0 - Root,
        write("node(",S,Key,Attr,Val,Src,").").
        write("edge(",Root,S,u,Pref_flag,").").
        build_ptree(Key, Attr, Root, Rest, false).
% most recent action was a confirmation
build_ptree(Key, Attr, Root, [H|Rest], Pref_flag) :-
        history(H,_,_,Val,c,_,_,_,Src),
        S is 0 - Root,
        write("node(",S,Key,Attr,Val,Src,").").
        write("edge(",Root,S,u,Pref_flag,")."
        build_ptree(Key,Attr,Root,Rest,Pref_flag).
% most recent action was an attribute resolution
build_ptree(Key, Attr, Root, [H|Rest], Pref_flag) :-
        history(H,_,_,_,a,Pref, LeftP, RightP, _),
        make_ptree(Key, LeftP, NewRoot1),
        setpref(Pref_flag, Pref, left, P),
        write("edge(",Root,NewRoot1, a, P,").").
        make_ptree(Key, RightP, NewRoot2),
        setpref(Pref_flag, Pref, right, Q),
        write("edge(",Root,NewRoot2, a, Q,").").
        build_ptree(Key, Attr, Root, Rest, false),!.
% most recent action was an entity resolution
build_ptree(Key, Attr, Root, [H|Rest], Pref_flag) :-
        history(H,_,_,_,e,Pref, LeftP, RightP, _),
        make_ptree(LeftP, Attr, NewRoot1),
        setpref(Pref_flag, Pref, left, P),
        write("edge(",Root,NewRoot1,e,P,").").
        make_ptree(RightP, Attr, NewRoot2),
        setpref(Pref_flag, Pref, right, Q),
        write("edge(",Root,NewRoot2,e,Q,").").
        build_ptree(Key, Attr, Root, Rest, false),!.
% base condition - empty history list
build_ptree(_,_,_,[],_).
```

Figure 2. Provenance tree construction algorithm

The provenance tree generated for the attribute value "Injured" for incident 104 in the final data table of Section 2 is shown in Figure 1. Nodes are labelled for clarity in this example with a row key (IncidID in

this case) and a column name to indicate the data value represented. The tree shows that the data value of interest has value 6, which was inherited from ancestor {104, wounded} and preferred over an alternate value in ancestor {104, cas} during attribute resolution of attributes "cas" and "wounded" into the new attribute named "injured". The tree also shows that this value was confirmed by user input. The value at {104, cas} is shown to be inherited from earlier values at {102, cas} and {104, cas} during entity resolution of incidents 102 and 104 into a new incident ID that the user chose to also call 104. In this case, both values were the same, so both edges are preferred. The value from {102, cas} is shown in the tree to have its origin in an update made by the user from a source document called "document 3".

We have defined a set of predicates over the Node and Edge tables to assist in answering common queries. We omit their Datalog definitions and define them informally here:

- view_origins – returns the list of node numbers of leaf nodes connected to the root node by preferred edges
- view_sources – returns the names of data sources from all leaf nodes connected to the root by preferred edges
- view_all_predecessors – returns the list of node numbers of all leaf nodes in the tree
- view_support – returns a count of the number of non-root nodes with the same data value as the root
- view_attribnames – returns the list of attribute names found in the tree nodes
- view_entityIDs – returns the list of entity (row) identifiers found in the tree nodes

Using these predicates, relational algebra, and the provenance graph for a selected data value, we can answer the kind of questions described in Section 2. To answer the question, *"How did we find out the date of incident 105?"* we build the provenance graph for the "Date" column of the row for incident 105, and evaluate the query view_sources(). We answer, *"Which incidents were combined from others in the table?"* using relational algebra over the history and data tables:

$$\pi_{\text{IncidID}}\,(\sigma_{\text{Row.Visible=True,Action=E}}\,(h \bowtie r))$$

The join in this expression produces a tuple for every entity that has a history table entry associated with one of its data values. The selection operator chooses those where the history table entry represents an entity resolution, and where the entity is still visible to the user.

The question, *"For each incident merged from redundant incidents, what were the incident IDs of the two predecessor rows we chose to resolve?"* can be answered by

$$\pi_{\text{merged, original-1,original-2}}$$
$$((\rho_{\text{incidID} \rightarrow \text{original-1}}\,(\pi_{\text{merged, IncidID}}\,((\rho_{\text{incidID} \rightarrow \text{merged}}\,(\pi_{\text{incidID,LeftParent}}(\sigma_{\text{Row.Visible=True, Action=E}}\,(h \bowtie r))))\bowtie_{\text{LeftParent = KeyVal}} r\,)))$$
$$\bowtie_{\text{merged}}$$
$$(\rho_{\text{incidID} \rightarrow \text{original-2}}\,(\pi_{\text{merged, IncidID}}\,((\rho_{\text{incidID} \rightarrow \text{merged}}\,(\pi_{\text{IncidID,RightParent}}(\sigma_{\text{Row.Visible=True, Action=E}}\,(h \bowtie r))))\bowtie_{\text{RightParent = KeyVal}} r\,))))$$

To answer the question, *"How many information sources have we found to support the casualty count for the IED incident in row 102?"* we can apply the view_origins() predicate to the provenance graph for the data value "Injured" in the row for incident 102. *"What other names did column Long in this table have in earlier versions of the data you gathered?"* can be answered by building a provenance tree for each data value in column Long, and taking the Union of view_attribnames() on each provenance graph. *"What information about incident 109, if any, was derived differently than the rest?"* can be answered by using make_ptree() to construct provenance trees for each data value in the selected row, then computing the tree edit distance [8] for each possible pairing of these values, and looking at the distribution of edit distances.

Provenance queries can also be written against the history table using the recursion capabilities supported in SQL:1999. As an example, to answer the question, *"Where did we find the date of incident 105?"*, one can write the following query, implicitly selecting the row identifier of the row for incident 105 (called X in this example) via a user interface.

```
WITH RECURSIVE
Value_Events(KeyVal, AttrName, Action, Preferred,
                LeftParent, RightParent, Source)
AS
(SELECT KeyVal, AttrName, Action, Preferred, Left
        Parent, RightParent, Source
 FROM History h
 WHERE h.KeyVal = X
        AND  h.AttrName = "Date"
        AND  h.SeqNum =
                (SELECT MAX (Seqnum)
                 FROM History h
                 WHERE h.KeyVal = X
                 AND h.AttrName = "Date"
                 AND h.Action != "Confirm"))
```

```
UNION
(SELECT KeyVal, AttrName, Action, Preferred,
        LeftParent, RightParent, Source
 FROM History h, Value_Events v1
 WHERE (h.KeyVal = v1.LeftParent
          AND v1.Preferred = Left
          AND h.AttrName = v1.AttrName
          AND h.Action = E) OR
        (h.KeyVal = v1.RightParent
          AND v1.Preferred = Right
          AND h.AttrName = v1.AttrName
          AND h.Action = E)  OR
        (h.AttrName = v1.LeftParent
          AND v1.Preferred = Left
          AND h.KeyVal = v1.KeyVal
          AND h.Action = A) OR
        (h.AttrName = v1.RightParent
          AND v1.Preferred = Right
          AND h.KeyVal = v1.KeyVal
          AND h.Action = A))

SELECT Source
FROM Value_Events v2
WHERE Source != ""
```

6. Overhead of Provenance Storage

Provenance overhead grows with the number of user actions. If a dataset is constructed using individual copy-and-paste actions, then the history table grows by one entry per data value pasted into the data table, plus entries for each update, confirmation, and resolution later affecting that data value. Let N be the number of data values pasted by the user, r the average number of updates made to each value, and c the average number of confirmations applied to each value. Note that the number of resolutions cannot exceed N - 1, because once a pair of values is resolved to form a new value, the original pair becomes non-visible and cannot be resolved further. The number of history table entries may be as large as

N	(due to initial population)
+ N * r	(due to revisions)
+ N * c	(due to confirmations)
+ N - 1	(due to resolutions)

Then the number of history table entries is less than

$$(2 + r + c)\ N$$

Revisions (r) and confirmations (c) per data value are potentially unlimited, but in practice, we expect that each data value will only be revised or confirmed a small number of times, and that number is independent of the total number of data values in the table. The overhead for storing provenance also must include a factor for the size of each history entry relative to the average size of each data value. The size of a history entry is constant in our implementation, and if we assume that the average size of a data value is also a constant, we can express the ratio of these as a constant k_h. This lets us express a bound on the overhead for storing provenance as

$$k_h * ((2 + r + c)\ N), \text{ or } O(N)$$

7. Related Work

Buneman, Chapman, Cheney, and Vansummeren [10,11] address provenance for manually curated data in scientific disciplines. Their work shares many of the same goals as ours. They address two forms of provenance ("where" and "why" [17]), but do not seem to address the ability to reconstruct the history of actions affecting data values ("what"). However, if historical data values and actions were included in their provenance tables, the expressive power of our approach and theirs, with regard to updates, would be the same. Their work addresses a limited set of user actions: copy-and-paste, insertion, and deletion (all of which we call updates). Our user action model includes entity resolution, attribute resolution (schema modification), and confirmation, though it does not at present include deletions.

Substantial work has been reported on computing provenance of data exchanged between structured, distributed scientific data stores. Orchestra [4], a prototype Collaborative Data Sharing System (CDSS), provides a general-purpose platform for data sharing between structured data sources maintained by distinct collaborating teams. Orchestra employs a detailed provenance model based on polynomials in semi-rings [12], where each term in a polynomial describes a partial mapping from source relations to target relations. The polynomial approach allows Orchestra to track several ancestors for one tuple, much like CHIME does for individual data values. As a result, Orchestra and CHIME support trust evaluation based on whether a user trusts the various sources described in the provenance polynomial. Orchestra also differs from our work. First, Orchestra computes provenance on a per-tuple basis, while we compute provenance on a per-value basis. Second, Orchestra focuses on only mappings expressed at the schema level, while our work focuses on mappings expressed at the individual data-

value level. In Orchestra, provenance data is computed when explicit "update exchange" events are issued. In our setting, we continuously record the history of user actions, and at any time allow the user to construct from this history a provenance tree to support useful queries. We also have a richer model for user confirmation, where the user can explicitly confirm a value; Orchestra has a simpler negative confirmation at the tuple level: a tuple that originated from another site, once deleted locally, will not be imported again.

Uncertainty-Lineage Databases (ULDBs) [13] store the lineage of database tuples along with data. A lineage function λ is defined for each relational operator available in the database, and lineage for each tuple is computed using the appropriate lineage functions each time an operation is performed. Both CHIME and ULDBs rely on well-behaved lineage: For example, lineage computation must be free of cycles. CHIME prevents cycles in lineage because the set of user actions on a data value is by definition cycle-free. Our work differs from the ULDB approach in that the source data for our provenance calculations are stored as a history table, which also enables other functionality, while ULDBs appear to store only lineage. Our work also differs in that CHIME retains provenance on a per-value basis rather than per tuple.

AutoMed [14] defines the notion of schema-transformation pathways that express data transformation and integration in a relational (data warehouse) environment. In AutoMed, transformation pathways are evolved dynamically to record the evolution of the warehouse schema. Lineage tracing of tuples may be derived from this pathway data. Our work is similar, but at the granularity of data values rather than tuples. We retain a history table that resembles the transformation pathways of AutoMed, and use it to compute provenance of individual data values, much as AutoMed does for tuples.

8. Conclusions and Future Work

We have defined a new data-management setting distinct from traditional information integration (including update exchange). We have shown how the history of user actions in our setting defines the Where, What, and Why provenance of the integrated datasets users create in that setting. Whether by automated or manual means, our data curation setting permits the provenance of sets of data values, including data from query answers or views, to be captured as long as the collective gathering work can be expressed in a set of value-at-a-time actions.

Using this model, we have shown how to answer user questions about data provenance. To test our ideas, we have implemented a prototype version of CHIME, supporting copy-and-paste, drag-drop, entity resolution, and search-browse with a Windows user interface; a Prolog implementation supporting all user actions defined here (update, confirmation, entity resolution, attribute resolution, and de-resolution), that creates the user data table and history table; and a Prolog implementation for creation and querying of provenance trees from history tables, as described in this paper.

At present, the CHIME data model is limited to first normal form. That is, though an attribute value may have a history of several values, at any point in time it has exactly one value. As Halevy, Franklin, and Maier point out, a realistic dataspace model must represent inconsistent and uncertain attribute values [2]. We are currently extending this work to cover representation and manipulation of such non-first normal form structures, incorporating some of the ideas of Jaeschke and Schek [15] and Arisawa, Moriya, and Miura [16].

CHIME provides integration that takes advantage of the high-quality decisions derivable from direct user actions. This trade-off in favor of high quality necessitates relaxing integration throughput. Other approaches, such as Orchestra [4], favor enabling high throughput by automated execution of a set of pre-constructed TGDs against incoming data sets. We plan to explore blending these approaches.

Acknowledgments

This work was supported in part by NSF grant IIS-0534762 and by DARPA.

References

[1] Halevy, A., Franklin, M., and Maier, D. "Principles of dataspace systems," In *Proceedings of the Twenty-Fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (PODS '06). ACM, 2006.

[2] Fagin, R., Kolaitis, P. G., Miller, R. J., and Popa, L. "Data exchange: semantics and query answering," *Theoretical Computer Science* 336, 2005.

[3] Green, T. J., Karvounarakis, G., Ives, Z. G., Tannen, V. "Update exchange with mappings and provenance," In *Proceedings of the 33rd international Conference on Very Large Data Bases* (VLDB '07). VLDB Endowment, 2007.

[4] Green, T., Karvounarakis, G., Taylor, N., Biton, O., Ives, Z., Tannen, V. "ORCHESTRA: facilitating collaborative data sharing," In *Proceedings of the 2007 ACM SIGMOD international Conference on Management of Data* (SIGMOD '07). ACM, 2007.

[5] Green, T., Karvounarakis, G., Ives, Z., Tannen, V. "Update exchange with mappings and provenance," In *Proceedings of the 33rd international Conference on Very Large Data Bases* (VLDB'07). VLDB Endowment, 2007.

[6] Archer, D., Delcambre, L. "Capturing Users' Everyday, Implicit Information Integration Decisions," *Conferences in Research and Practice in Information Technology* vol. 83. Auckland, NZ, 2007.

[7] Archer, D., Delcambre, L. "Definition and Formalization of Entity Resolution Functions for Everyday Information Integration," In *Proceedings of SDKB 2008*, Nantes, France, 2008.

[8] Bille, P. "A survey on tree edit distance and related problems"*Theoretical Computer Science* 337, 2005.

[9] Abiteboul, S. and Duschka, O. M. "Complexity of answering queries using materialized views," In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (PODS '98). ACM, 1998.

[10] P. Buneman, A. Chapman, J. Cheney, and S. Vansummeren. "A provenance model for manually curated data," In *Proceedings of the International Provenance and Annotation Workshop* (IPAW'06), 2006.

[11] Buneman, P., Chapman, A., and Cheney, J. "Provenance management in curated databases," In *Proceedings of the 2006 ACM SIGMOD international Conference on Management of Data* (SIGMOD '06). ACM, 2006.

[12] Green, T., Karvounarakis, G., Tannen, V. "Provenance semirings," In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (PODS '07). ACM, 2007.

[13] Benjelloun, O., Das Sarma, A., Halevy, A., Theobald, M., Widom, J. 2008. "Databases with uncertainty and lineage," *VLDB Journal* 17, 2, 2008.

[14] Fan, Hao, Poulovassilis, A. "Using schema transformation pathways for data lineage tracing," *Lecture Notes in Computer Science* 3567, 2005.

[15] Jaeschke, G., Schek, H. "Remarks on the Algebra of Non First Normal Form Relations," In *Proceedings of the 1st ACM SIGACT-SIGMOD Symposium on Principles of Database Systems* (PODS '82), ACM, 1982.

[16] Arisawa, H., Moriya, K., and Miura, T. "Operations and Properties on Non-First-Normal-Form Relational Databases," In *Proceedings of the 9th international Conference on Very Large Data Bases* (VLDB '83) VLDB Endowment, 1983.

[17] Buneman, P., Khanna, S., Tan, W. C. "Why and Where: A Characterization of Data Provenance," In *Proceedings of the 8th International Conference on Database Theory* (2001). J. V. Bussche and V. Vianu, Eds. Lecture Notes In Computer Science, vol. 1973. Springer-Verlag, London, 2001.

[18] Buneman, P., Cheney, J., Tan, W. C., Vansummeren, S. "Curated Databases," In *Proceedings of the 27th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems* (PODS'08), ACM, 2008.

[19] Buneman, P. "Curated Databases," In *Proceedings of the 4th International Conference on Web Information Systems Engineering* (WISE'03), 2003.