Senbazuru: A Prototype Spreadsheet Database Management System

Zhe Chen  Michael Cafarella  Jun Chen  Daniel Prevo  Junfeng Zhuang

University of Michigan
Ann Arbor, MI 48109-2121
{chenzhe, michjc, chjun, drpre, jimmyzhu}@umich.edu

ABSTRACT

Spreadsheets have become a critical data management tool, but they lack explicit relational metadata, making it difficult to join or integrate data across multiple spreadsheets. Because spreadsheet data are widely available on a huge range of topics, a tool that allows easy spreadsheet integration would be hugely beneficial for a variety of users.

We demonstrate that Senbazuru, a prototype spreadsheet database management system (SSDBMS), is able to extract relational information from spreadsheets. By doing so, it opens up opportunities for integration among spreadsheets and with other relational sources. Senbazuru allows users to search for relevant spreadsheets in a large corpus, probabilistically constructs a relational version of the data, and offers several relational operations over the resulting extracted data (including joins to other spreadsheet data). Our demonstration is available on two clients: a JavaScript-rich Web site and a touch interface on the iPad. During the demo, Senbazuru will allow VLDB participants to search spreadsheets, extract relational data from them, and apply relational operators such as select and join.

1. INTRODUCTION

Spreadsheets are an extremely popular data management tool, allowing users to complete a range of data tasks commonly associated with relational systems: projection, sorting, aggregation, and simple ETL (Extract, Transform and Load) jobs. The sheer number and diversity of spreadsheet sheets is remarkable; our Web crawl using the URLs in the ClueWeb09 corpus [4] found 410,554 spreadsheet files scattered across 51,252 domains. Moreover, the data is often the result of intense human focus and effort. These spreadsheets should be prime targets for ad-hoc data integration and analysis. For example,

Policy expert Fred wants to see whether the strength of the connection between smoking and lung cancer is consistent across all U.S. states. Fred does not have the relevant data at hand, but assumes it is “out there” on the Web somewhere. In one sense, the user is fortunate: different branches of the government have collected the data relevant to his task and made them available online, likely via two separate downloadable spreadsheets, one for the smoking statistics and one for the lung cancer statistics. Unfortunately, finding such data via current search engines is quite tedious. In our case, Fred would need to issue a text query, then review all the returned documents before finding the relevant spreadsheets. Moreover, because spreadsheets do not have explicit relational schema, the user cannot benefit from society’s huge investment in data integration tools that work on relational databases. Instead, Fred will likely have to write custom code to combine the two spreadsheets.

Extracting relational data from spreadsheets would enable traditional database management tools, such as data integration systems, to be applied to spreadsheet data. There is a body of recent work that performs database-style operations inside a spreadsheet interface [6, 8, 11], but these approaches are not useful for spreadsheet datasets that already exist. Other studies have attempted to transform spreadsheet data into the relational model, making further integration among spreadsheets possible. Some extraction approaches are not useful for spreadsheet datasets that already exist. Other studies have attempted to transform spreadsheet data into the relational model, making further integration among spreadsheets possible. Some extraction approaches are not useful for spreadsheet datasets that already exist. Other studies have attempted to transform spreadsheet data into the relational model, making further integration among spreadsheets possible. Some extraction approaches are not useful for spreadsheet datasets that already exist. Other studies have attempted to transform spreadsheet data into the relational model, making further integration among spreadsheets possible.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were invited to present their results at The 39th International Conference on Very Large Data Bases, August 26th - 30th 2013, Riva del Garda, Trento, Italy. Proceedings of the VLDB Endowment, Vol. 6, No. 12
Copyright 2013 VLDB Endowment 2150-8097/13/10.. $ 10.00..

1 The U.S. Census Bureau: http://www.census.gov/
which cells are values, or which attributes describe which values. Also, the spreadsheet does not clearly state how to assemble the attributes into coherent sets. For example, the left-hand attributes in rows 26 and 32 belong to the same set (race), but row 25 does not. This grouping information is critical when constructing any relational format of the data. In summary, Figure 1 shows a clean, high-quality spreadsheet, recovering relational data from it requires us to: (1) detect attributes and values; (2) identify the hierarchical structure of left and top attributes if any exists; (3) generate relational tuples; and (4) assemble the tuples into high-quality relational tables.

- Repair – Extracting metadata is brittle: even a single extraction error can yield wrong tuples that will poison any downstream applications. For example, if the extractor fails to recognize that row 20 annotates row 26, the extracted relational tuples from row 27 to 31 will all be incorrect, as none contains any mention of the attribute Male. Thus, it should always be possible to manually repair any automatic extraction errors.

We demonstrate that Senbazuru, a prototype spreadsheet database management system (SSDBMS), is able to extract relational information from a large number of spreadsheets; doing so opens up opportunities for data integration among spreadsheets and with other relational sources. Senbazuru consists of three functional components we view as critical for a useful SSDBMS:

- Search – Using a textual search-and-rank interface, the search component allows a user to quickly locate relevant datasets in a huge Web spreadsheet corpus.
- Extract – The extract component is composed of a background extraction pipeline that automatically obtains relational data from spreadsheets, and a repair interface that allows users to manually repair extraction errors. Moreover, this component automatically exploits commonalities among errors to probabilistically reapply one user fix to other similar mistakes, thereby minimizing explicit manual intervention.
- Query – The query component supports basic relational operators, especially selection and join, which the user can apply to spreadsheet-derived relations.

We have a working prototype of Senbazuru, available as a desktop Web application and also as an iPad application. Recall our policy expert Fred, now newly equipped with an SSDBMS. Fred can use Senbazuru’s search component to rank datasets by relevance to his text query, smoking by state. He then picks the best hit. The extract component automatically generates a relational table for the spreadsheet, allowing Fred to fix an extraction error with a single drag-and-drop. Fred can also use the query component to select a portion of the data about smoking by states, and join with another dataset about cancer by states. Overall, Senbazuru requires only a few clicks and two text queries, and yields a new dataset that is exactly what Fred wants.

In the rest of this paper, we will give an overview of the backend framework of Senbazuru (Section 2), describe how the end-user can interact with Senbazuru’s interface during the demonstration with a walk-through example (Section 3), and conclude with a brief summary of technical problems the system addresses (Section 4).

Figure 1: A screenshot of a spreadsheet with implicit hierarchical attributes in its left-hand column. At bottom, the relational tuple corresponds to the highlighted value 13.7.

2. SYSTEM FRAMEWORK

In this section, we describe the backend software framework that allows Senbazuru to offer the standard SSDBMS services: search, extract, and query, as shown in Figure 2.

Search – The search component helps the user to retrieve relevant datasets in a potentially enormous corpus via Web search-style relevance ranking. Currently, Senbazuru has indexed more than 1,800 spreadsheets collected from the U.S. Census Bureau. For each spreadsheet in the corpus, the indexer uses the Python xlr package to extract text from each cell, then uses Apache Lucene to index the text. When a query arrives, the searcher uses the inverted index and TFIDF-style ranking to sort the datasets by relevance. As with a standard Web search, the results of search comprise a list of potentially useful objects for the user to examine.

Extract – The extract component consists of a sequence of pipeline components that convert the data in each spreadsheet into the relational model. Most of the extract takes place when the corpus of spreadsheets is first added to the system, before any query arrives. A small portion of the extract is deferred until the user provides manual error repairs.

1. Frame Finder – First, each raw spreadsheet is sent to the frame finder. This module identifies data frame structures in each spreadsheet. A characteristic data frame layout is shown in Figure 1: a rectangular value region of numeric values (outlined with the dashed rectangle), with additional rectangular attribute regions above (labeled top attribute and outlined with a solid rectangle) and to the left (labeled left attribute, similarly outlined). We use a conditional random field (CRF) [9] to assign a semantic label to each non-empty row in the spreadsheet, thus finding the data frame. The module passes the data frame to the next step and discards everything else in the raw spreadsheet.

2. Hierarchy Extractor – The next stage is the hierarchy extractor, which recovers the attribute hierarchies for the data frame’s attribute regions. In each attribute region, we want to know which attributes describe which other attributes. For example, in Figure 1, the attribute at row 31
3. DEMONSTRATION

We plan to show an end-to-end demonstration of Senbazuru to search, extract, and query spreadsheets. Users can choose to interact with Senbazuru using either the Web browser client or the iPad tablet client. In this section, we first describe how to use Senbazuru’s interface, then show a walk-through example.

3.1 User Interface

The two client interfaces for Senbazuru are shown in Figure 4. They allow users to search for data, to view and edit the extract results, and to use query operations.

Search – As with other search-and-rank tools, a user types keywords in the search box, and obtains a list of relevant spreadsheets. She can then browse results and select the most relevant one. For example, in Figure 4 (a), a user enters “smoking” as the search query. She can examine the top hit’s raw spreadsheet by clicking “Spreadsheet” or check other relevant spreadsheets by clicking “Next.”

Extract – After selecting the most relevant spreadsheet, a user can use extract to transform the spreadsheet data into a relational table. She can review the extracted hierarchical metadata by clicking “Data Tree.” Figure 3 shows the interface for viewing the extracted hierarchies. To repair any extraction errors, she can drag and move a node of the tree from one place to another. After observing the user’s repair action, Senbazuru will automatically re-run extract and display a new tree. For example, as shown on the left of Figure 3, a user performs a repair by clicking and dragging White, total so that it becomes a child of Male, total. Meanwhile, the extract component automatically discovers that Black, total should also be moved. Thus, the user’s single repair action can trigger multiple fixes, yielding the hierarchy shown on the right. After repairing extraction errors, the user can review the generated relations by clicking “Relational Table,” as shown in Figure 4(a).

Query – Users can perform query operations on the extracted relational table. They are not required to write SQL statements and can apply select and join via the interface:

1. Select – The select feature, also called filter, is similar to executing a selection query on the recovered relation. Clicking “Filter,” a user can use a faceted-search interface to specify the filter conditions. This interface is automatically composed by Senbazuru. For example, Figure 4 (a) shows that the user limits the displayed data to smoking statistics for people who are Female, Black, and 18 to 24 years old.

2. Join – The join feature allows users to integrate two arbitrary spreadsheet-derived relations. The user starts by

Figure 3: Senbazuru recovers the attribute hierarchy seen in Figure 1, and makes the recovered structure available for manual user repairs.
applying the search and extract features as described above.

Once the user has found a good result, she clicks “Join” and enters a second text query. The query yields a second ranked result list. She chooses a relevant join target from this ranked list and obtains a correctly extracted relation. She indicates which columns from each dataset should be used as an equijoin key.

For example, Figure 4 (b) shows a screenshot of the join process on our iPad client. When the user touches a column, Senbazuru highlights it in bright yellow, as shown on the left of the figure. Meanwhile, Senbazuru faintly highlights a column on the right-hand table to indicate a possible join. The user can drag the column, highlighted in bright yellow, from the left-hand relation to the right and release it. This action indicates the join key; Senbazuru executes the appropriate join and shows the user the resulting brand-new relation.

3.2 A Walk-through Example

We will demonstrate Senbazuru’s workflow through Fred’s example. Recall that our policy expert Fred wants to find the relationship between smoking and lung cancer across the U.S. states. Using Senbazuru, Fred can obtain the requested data through the following steps:

1. Search for “smoking by state.”
2. Browse the returned spreadsheets and select the most relevant one.
3. Click “Data Tree” and check the correctness of the extracted attribute hierarchies.
4. Repair the hierarchies if any extraction errors exist.
5. Click “Data Table” to review the extracted relation.
6. Repeat the process from step 1 to get the relational table for the most relevant lung cancer spreadsheet by querying “lung cancer by state.”
7. Specify join columns to create a new table.
8. Review the result, using the faceted select interface.

In addition to Fred’s example, VLDB attendees are welcome to try many other interesting queries, such as “employment statistics 2010” and “Michigan GDP.”

4. CONCLUSION

Senbazuru is a prototype spreadsheet management system. It searches a large number of Web crawl spreadsheets, and it automatically transforms spreadsheets into relations while allowing users to fix the extraction errors effectively and efficiently. Finally, it supports selection queries on the resulting relations and join queries to integrate arbitrary spreadsheets. We believe a demo of Senbazuru will be of great interest to the VLDB attendees and the database community as a whole.

5. ACKNOWLEDGMENTS

This project is supported by National Science Foundation grants IIS-1054913 and IIS-1064606, as well as gifts from Dow Chemical, Yahoo!, and Google.

6. REFERENCES