Annotating Database Schemas to Help Enterprise Search

Eli Cortez, Philip A. Bernstein, Yeye He, Lev Novik

Microsoft Corporation
Redmond, WA, 98052, U.S.A

{eli.cortez, philbe, yeyehe, levn}@microsoft.com

ABSTRACT

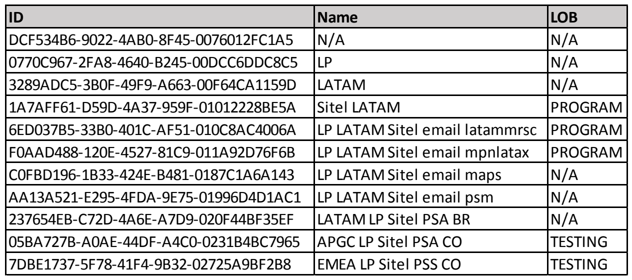
In large enterprises, data discovery is a common problem faced by users who need to find relevant information in relational databases. In this scenario, schema annotation is a useful tool to enrich a database schema with descriptive keywords. In this paper, we demonstrate Barcelos, a system that automatically annotates corporate databases. Unlike existing annotation approaches that use Web oriented knowledge bases, Barcelos mines enterprise spreadsheets to find candidate annotations. Our experimental evaluation shows that Barcelos produces high quality annotations; the top-5 have an average precision of 87%.

1. INTRODUCTION

Large enterprises typically have thousands of relational databases, each containing tens to hundreds of tables, with many columns per table. To generate reports or new applications, users face the problem of *data discovery.* They have to find database tables that are potentially relevant to the task at hand. Then, for each of these candidate tables, they need to understand its content to determine whether it is truly relevant. Both of these steps are quite challenging. A user is typically familiar with only a small fraction of the enterprise's databases. For the others, the schema's table and column names are often not very descriptive of the content. Therefore, even if there is a catalog of schemas covering the enterprise's databases, keyword search over schema information is ineffective at finding candidate tables and columns.

To illustrate the last point, we sampled 4216 data columns in 639 tables from 29 databases used by Microsoft's IT organization. We found that many frequently-used column names are very generic, such as: `name`, `id`, `description`, `field`, `code`, and `column` (representing **28**% of all columns we sampled). These generic column names are useless for helping users find tables that have the data they need.

The table shown in Figure 1 gives one such example. Notice that the first two column names in the table are overly-generic and uninformative. The last column name is an abbreviation for "line of business", which may be cryptic for some users. Such non-descriptive column names make it difficult to search and understand the table.



ID	Name	LOB
DCF534B6-9022-4AB0-8F45-0076012FC1A5	N/A	N/A
0770C967-2FA8-4640-B245-00DCC6DDC8C5	LP	N/A
3289ADC5-3B0F-49F9-A663-00F64CA1159D	LATAM	N/A
1A7AFF61-D59D-4A37-959F-01012228BE5A	Sitel LATAM	PROGRAM
6ED037B5-33B0-401C-AF51-010C8AC4006A	LP LATAM Sitel email latammrsc	PROGRAM
F0AAD488-120E-4527-81C9-011A92D76F6B	LP LATAM Sitel email mpnlatax	PROGRAM
C0FBD196-1B33-424E-B481-0187C1A6A143	LP LATAM Sitel email maps	N/A
AA13A521-E295-4FDA-9E75-01996D4D1AC1	LP LATAM Sitel email psm	N/A
237654EB-C72D-4A6E-A7D9-020F44BF35EF	LATAM LP Sitel PSA BR	N/A
05BA727B-A0AE-44DF-A4C0-0231B4BC7965	APGC LP Sitel PSA CO	TESTING
7DBE1737-5F78-41F4-9B32-02725A9BF2B8	EMEA LP Sitel PSS CO	TESTING

Figure 1: A typical corporate database table with generic column names (and redacted values).

Some companies address this problem by using *data stewards* to enrich database tables and columns with textual descriptions and keyword annotations. Data stewards also serve as consultants to help others find and understand the data they need. However, this approach is time consuming and hence expensive. Therefore, data stewards often focus only on the most valuable databases, ignoring databases that are less frequently used.

To help solve the preceding problems, we developed Barcelos, a system that automatically generates candidate keywords to annotate columns of database tables. This greatly reduces the effort of annotating database schemas, even if some human curation of the generated keywords is later required. Moreover, it enables all databases in the enterprise to be annotated, even those that are infrequently used.

Barcelos works by mining spreadsheets, which are typically abundant in an enterprise's intranet. Many of these spreadsheets were generated by queries over the enterprise's databases. Our main observation is that since spreadsheets are usually designed to be read, they often have more meaningful column names than those of the database columns from which the spreadsheet was generated. Therefore, we can use the spreadsheet's column names as candidate annotations for the corresponding database columns. The technique is quite effective. For the example in Figure 1, Barcelos produces annotations "TeamID" and "Team" for the first column, "Delivery Team" and "Team" for the second column, and "Line of Business" and "Business" for the third.

To the best of our knowledge, Barcelos is the first published system to automatically annotate proprietary corporate databases. We make the following contributions:

- A method to automatically extract tables from a corpus of enterprise spreadsheets.

Proceedings of the VLDB Endowment, Vol. 8, No. 12
Copyright 2015 VLDB Endowment 2150-8097/15/08.

• A method for identifying and ranking relevant column annotations, and an efficient technique for calculating it.

• An implementation of our method, and an experimental evaluation that shows its efficiency and effectiveness.

2. RELATED WORK

People have looked at a similar problem of discovering semantics of tables in the context of Web tables [5, 6, 8], where the goal is better Web table understanding to benefit Web table search. Solutions rely on general-purpose Web-oriented Knowledge Bases (KBs), such as YAGO [7] and FreeBase [1]. These KBs have rich entities and relationships, but require extensive effort to build and maintain [1, 7].

While these KB-based techniques are natural in the context of Web tables, they do not generally apply to enterprise table annotation. Given the proprietary nature of enterprise data, it is unlikely that internal data such as sales reports, organization hierarchies, or employee names can be found on the Web. Thus, they will be absent from existing Web-oriented KBs, making KB-based annotation unsuitable for enterprise data annotation. Furthermore, building KBs from scratch for each individual enterprise is unlikely to be worthwhile, given the amount of effort required and the number of applications that can benefit from it.

One schema-only approach for finding attribute synonyms in Web tables uses attribute-name co-occurrence [2]: if attributes A and B co-occur frequently and so do A and C, then B and C are likely to be synonyms. But this approach is not always robust, due to ambiguity in attribute names (e.g., "id", "code"). Instead, we use a value-based approach that finds synonyms by intersecting sets of values in attributes B and C. Data values are less ambiguous than column names, and the set of values in a column is indicative of the concept in question, making our technique more robust.

NLP-based techniques, such as textual Hearst patterns as used in Probase [9] and YAGO [7], also apply to the problem of table annotation. The idea is to use documents and text patterns to generate IsA relationships. For example, if we see enough text segments like "Microsoft products, such as Internet Explorer 10, SQL Server 2012, ...", we could generate IsA relationships: ("Internet Explorer 10", "Microsoft product"), ("SQL Server 2013", "Microsoft product"). We can use these IsA labels to annotate columns if enough labels in the same column agree. We tried this approach by instantiating Hearst patterns with common instance values and using them as text queries in Microsoft's internal search engine. Unfortunately, we could not retrieve even a single document for many popular IsA pairs that we initially expected to generate many hits. We believe the reason is that enterprise documents are much sparser than Web documents, perhaps because people are not as incentivized to create textual content in an enterprise setting as in the Web. Instead, they may spend more time crunching numbers and putting together reports, which end up becoming structured data such as spreadsheets. Our spreadsheet-based technique thus adapts to the characteristics of enterprise data and leverages this unique enterprise data asset for annotation.

3. SYSTEM ARCHITECTURE

Figure 2 presents a high level picture of Barcelos' architecture. The offline phase (the lower half) consists of the spreadsheet crawling and indexing components. The online phase (the upper half) implements an annotation server

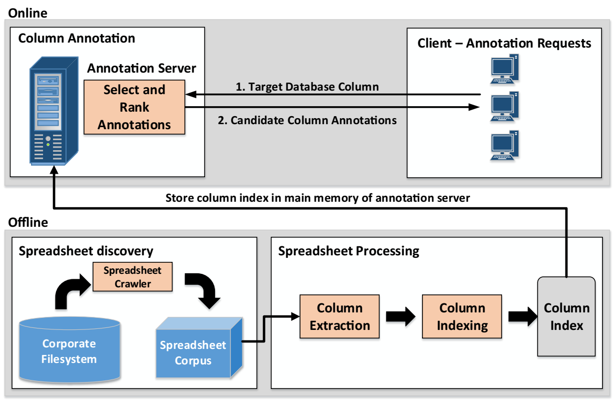


Figure 2: End-to-End system architecture.

that selects and ranks relevant annotations for each target database column. We briefly describe each component before going into detail.

• Spreadsheet discovery: This component crawls an enterprise intranet for spreadsheet files. Spreadsheet files can also be obtained without crawling by accessing the indexes of intranet search engines, such as Microsoft SharePoint and Google Search Appliance.

• Spreadsheet Processing: This component has two steps. The first step extracts tables from spreadsheets, where identifying table boundaries and header rows are the key challenges. The second step builds indexes using column headers and values, to speed up the annotation phase. The indexes map columns to values and vice versa.

• Column Annotation: We calculate a similarity score to rank candidate annotations for a given database column. The score is based on value overlap between the database and spreadsheet columns plus other contextual information.

3.1 Spreadsheet Discovery

The goal of this component is to create a spreadsheet corpus that represents the data within the enterprise. Its implementation depends on where the files are stored and how they are protected. It could be as simple as traversing directories, or can be more sophisticated by crawling intranet pages and dealing with privacy and file permissions.

3.2 Spreadsheet Processing

3.2.1 Table extraction from spreadsheets

Extracting tabular data is a challenging task by itself. Spreadsheet files usually contain multiple worksheets, each potentially containing multiple tables. Unlike tables in HTML files [2], worksheets have no standard delimiters between tables. We therefore used heuristics to guide the extraction task. For example, visual borders are a very strong signal of a delimited area of spreadsheet cells, and font formatting (bold, italic) in row-one indicates a possible table header.

Considering the content information (strings) and the formatting style (font color, borders, cell color), we extract the tables in a worksheet by finding the coordinates that represent a squared area, called the *content area*. It is the largest area within a worksheet that covers all textual content and is identified by the coordinates of its four corners.

If the content area has no empty cells, it is likely to be the one and only table in the worksheet. If it does have empty cells, the worksheet might contain multiple tables. To find them, we iterate over the cells in the content area, top-down and left-right. When an empty row and empty column are found, we assume they comprise a table boundary and restart the iterative process looking for new tables.

In order to extract column names from table header rows, we use a rule-based approach to detect header rows (e.g., if the first row has non-text fields, then it is unlikely to be a header row). Existing techniques such as classifiers used in [3] can also be applied here. We further use a rule-based classifier to determine the data-type of each column.

Spreadsheet columns whose names are not likely to be good annotations are discarded. In particular, we remove tables that have only one column and spreadsheet columns where more than 70% of the cells are empty.

In our experiments, this simple solution does an excellent job extracting tables from a diverse set of spreadsheet files. For ease of manipulation, each extracted table is stored as a JSON file and its schema (i.e., column names and data types) is stored in a SQL database.

3.2.2 An inverted value index for efficient processing

Depending on the size of the spreadsheet corpus, the number of extracted tables and columns can be in the millions. As we will see, the search for column annotations involves a set-similarity calculation over values in spreadsheet columns. To speed this up, we build two main-memory indexes. One of them maps column-value→column-id, and the other maps column-id→column-value. These are reminiscent of classical inverted indexes used for information retrieval, and are amenable to sort-merge style intersection joins.

We considered other indexing schemes for estimating set-similarity, for example, locality sensitive hashing schemes such as min-hashing for Jaccard similarity. As we will see, our scenario requires the use of Jaccard Containment, for which min-hash does not apply. While there are other techniques for Jaccard Containment, such as prefix filters [4], they generally require a fixed similarity threshold to be known *a priori*.

3.3 Column Annotation

For a given target database column c, the annotation phase returns a rank-ordered list of possible annotations for c. Intuitively, the name of a spreadsheet column sc is a valid candidate if its values overlap those of c. In the next section we describe our approach of ranking candidate annotations.

3.3.1 A regression model to rank annotations

We built a hand-tuned regression model to rank candidate annotations, mainly based on two groups of features. The first group are value-related, denoted by VR_f, which measure the set-similarity between values in c and sc. The larger the overlap between the sets of values, the more likely that the column name for sc is also a good name for c.

Symmetric set-similarity measures such as Jaccard Similarity are not suitable in our case, because the relationship between c and sc is in fact asymmetric. If c is mostly contained in sc, then the name of sc is very likely a good name for c. However, if sc is mostly contained in c, it is not guaranteed that the name of sc applies to c, because sc might be just a sub-concept of c. For example, sc may list all US sales districts, while c may have all sales districts in the world. In this case, the column name of sc, "US sales district," would not be a good annotation for c.

We therefore compute similarity using the Jaccard Containment [4] of sc in c, which is defined as:

$$JC(sc, c) = \frac{|V(sc) \cap V(c)|}{|V(c)|} \qquad (1)$$

where $V(sc)$ and $V(c)$ are the sets of values in sc and c, respectively. In general the value set can be weighted by TF-IDF-like weighting schemes. However, we find that values in enterprise database columns are usually quite unique and distinct, and the distribution of column values are not as skewed as (say) stop-words in general text processing. We therefore use uniform weighting in our model.

In order to measure $VR_f(sc, c)$, we use a weighted combination of $JC(c, sc)$ and $JC(sc, c)$ as follows.

$$VR_f(sc, c) = \beta\ JC(sc, c)\ +\ (1 - \beta)\ JC(c, sc) \qquad (2)$$

We empirically tuned parameter β. Setting $\beta = 0.2$ achieves robust performance across the different databases we tested. This is consistent with our analysis above, that $JC(c, sc)$ should be given more weight because of the asymmetric relationship between c and sc. However, $JC(sc, c)$ is still significant; if a large fraction of values of sc is not in c, then sc's column name is probably not a good annotation for c.

It turned out that these value-based features alone are not always enough. For example, consider a database column with a small number of values. Even if it is mostly contained in a spreadsheet column sc, it may still refer to a very different concept than sc. To address this issue, Barcelos uses context information in other attribute names in the same table to better determine the concept in question.

This gives rise to the second group of features, denoted by CR_f, that uses the similarity of other attribute names in the database table and spreadsheet table. The intuition is that if attribute names of the two tables are similar, then chances are they refer to the same thing, making us more confident to use one column to annotate the other.

Let *context*(sc) and *context*(c) be the set of unique tokens in the contextual attribute names of sc and c, respectively. We use the standard Jaccard similarity for $CR_f(sc, c_i)$:

$$CR_f(sc, c_i) = Jaccard(context(sc), context(c_i)) \qquad (3)$$

Finally, the overall ranking score is a linear combination of these two components VR_f and CR_f.

$$Score(c|sc) = \alpha\ VR_f(sc, c_j)\ +\ (1 - \alpha)\ CR_f(sc, c_j) \qquad (4)$$

where α controls the relative importance of value-related and context-related features. With $\alpha = 0.7$, Barcelos makes robust ranking predictions across the databases that we tested, as we will see in the next section.

It is possible to also consider other features, such as the number of spreadsheet columns in the corpus supporting a particular annotation. For example, if column c overlaps with 1000 identical spreadsheet columns that have the same column name A, then A is likely to be a good annotation for c. Aggregating such signals using models like Noisy-Or should further improve our scoring method.

3.3.2 Experimental evaluation

To evaluate the proposed approach, we conducted experiments that assess the quality of the annotations provided by Barcelos. We ran the spreadsheet discovery phase on some Microsoft-internal SharePoint sites, which found over 500K spreadsheet files. Barcelos could open only 48K of them because it can only handle the latest Excel format (i.e., .xlsx) and many files were encrypted. It extracted 589K tables from those files, producing indexes over 1.1M columns, 24.4M values, and 97.2M value-occurrences, which we used for all of our experiments.

We first evaluated the quality achieved by Barcelos compared with the KB-based approach in [8]. To build the KB required by [8], we used our spreadsheet table corpus, treating each column name as a concept and each column-value as an instance of the concept. We randomly selected 20 database columns to receive annotations and computed the top-K annotations using Barcelos and using the scoring function in [8]. As shown in Table 1, the precision of Barcelos is uniformly better, on average by 17%.

Table 1: Comparing the top-K annotations of Barcelos and the KB-based approach of [8].

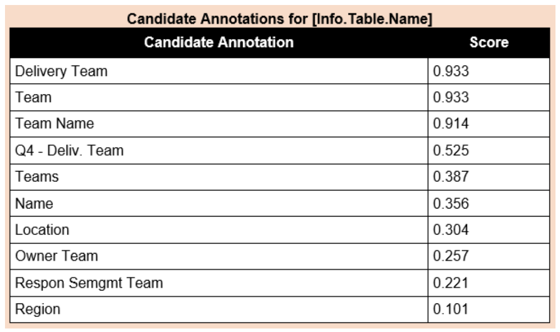
Metric	Barcelos	Baseline	Gain
Precision of top 1	**0.95**	0.80	19%
Precision of top 5	**0.89**	0.77	16%
Precision of top 10	**0.76**	0.65	17%

We also evaluated the precision of Barcelos to annotate 120 database columns from 30 corporate databases. The precision of the top annotation was again 95% and that of the five top-ranked annotations was 87%. These results show that our approach of mining enterprise spreadsheets for database column annotations yields high quality results. In our opinion, the precision is good enough to deploy in a commercial setting.

Finally, we evaluated the latency of the proposed system for annotating those 120 database columns. The average response time was less than one second per database column, making it suitable for interactive online annotation.

4. DEMONSTRATION DETAILS

We will demonstrate Barcelos on the spreadsheet corpus and database columns described in Section 3.3.2. Users can interact with the system by choosing a database column as input. The interface displays the table schema and column values of the selected column. Pressing "submit" tells Barcelos to generate a rank-ordered list of annotations with their scores. Figure 3 gives a screen-shot of the result displayed by Barcelos for the Name column in Figure 1.



Candidate Annotations for [Info.Table.Name]

Candidate Annotation	Score
Delivery Team	0.933
Team	0.933
Team Name	0.914
Q4 - Deliv. Team	0.525
Teams	0.387
Name	0.356
Location	0.304
Owner Team	0.257
Respon Semgmt Team	0.221
Region	0.101

Figure 3: Barcelos' output for the table in Figure 1.

5. CONCLUSION AND FUTURE WORK

We described a new system, Barcelos, that annotates database columns by mining enterprise spreadsheets. These annotations can be used to help users find relevant databases and understand their content. For a given database column *c*, Barcelos returns a rank-ordered list of names to use as annotations for *c*. It calculates this list using the names of spreadsheet columns whose data heavily overlaps that of *c*. Our experiments over a large spreadsheet corpus showed high precision for the top five names generated.

Our technique is general in that it can use metadata from any structured dataset (the "source") to annotate any other structured data (the "target"). For example, one could use it to annotate Hadoop data files, using spreadsheets as the source dataset. In fact, the source and target can be the same dataset. Given a field in the dataset, one could use the other fields as the source and thereby find the names of other fields that have highly similar data. For a large dataset, one probably needs to sample the data values, which might require modifying the similarity scoring used in Barcelos.

One extension worth exploring is to use SQL scripts and stored procedures. SQL queries often rename the result columns in human-friendly ways, e.g., with the "`SELECT Column-A AS Name-B`" construct. Given a query log or set of scripts and their associated databases, one could propagate these human-friendly column names back to databases. One difficulty with this approach is the lack of a centralized repository where all SQL scripts used in an enterprise can be obtained. (Spreadsheets are more easily obtainable in a centralized manner.) The question is whether this technique provides more or better annotations than the spreadsheet mining approach presented here.

6. REFERENCES

[1] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250, 2008.

[2] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *Proceedings of VLDB*, (1):538–549, 2008.

[3] M. J. Cafarella, J. Madhavan, and A. Y. Halevy. Web-scale extraction of structured data. *SIGMOD Record*, pages 55–61, 2008.

[4] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *Proceedings of ICDE*, page 5, 2006.

[5] J. Fan, M. Lu, B. C. Ooi, W.-C. Tan, and M. Zhang. A hybrid machine-crowdsourcing system for matching web tables. In *Proceedings of ICDE*, pages 976–987, 2014.

[6] R. Pimplikar and S. Sarawagi. Answering table queries on the web using column keywords. *Proceedings of the VLDB Endowment*, 5(10):908–919, 2012.

[7] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A core of semantic knowledge unifying wordnet and wikipedia. In *WWW*, pages 697–706, 2007.

[8] P. Venetis, A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, G. Miao, and C. Wu. Recovering semantics of tables on the web. *Proceedings of VLDB*, (9):528–538, 2011.

[9] W. Wu, H. Li, H. Wang, and K. Q. Zhu. Probase: A probabilistic taxonomy for text understanding. In *SIGMOD*, pages 481–492, 2012.