Sample Query Generation: Joins and Projection

Jason Ganzhorn
{ganzhj@cs.washington.edu}

11 March 2011

1 Introduction

This paper presents a component in a simple query generation algorithm which is reliant only on the schema and data of the data set generated queries are to be run on. The entire system is meant to form a pipeline for the generation of sample queries which can be studied by researchers and adapted to their purposes. The end goal is to produce queries which highlight interesting information about the database which the researcher might not have already thought of. The algorithm component developed here is a step along that path.

2 Motivation

The SQLShare[6] system is designed to allow users with little SQL experience to upload data and construct databases with said data. Modern experiments in many fields generate large amounts of data, which is often collected into spreadsheets and manually manipulated by the researcher in an attempt to find information which matches certain criteria. Such querying of the data is far more easily done by setting up a database and querying it. Unfortunately, traditional database setup and maintenance require technical expertise often out of the reach of a small research team. SQLShare aims to significantly ease database setup by allowing researchers to simply upload their data to a web service which automatically sets up the database and then provides a simple interface to query the data.

Unfortunately, it is still the case that a fairly extensive knowledge of SQL is required to fully utilize the power available in SQLShare. Having the data in a format such that it can easily be queried is arguably of little use to a researcher who does not know how to query it.

Researchers are generally very intelligent, however, which leads to the key hypothesis of our system. SQL has a generally clear and self-descriptive syntax, and therefore we believe that even if researchers do not know how to formulate SQL queries from scratch, they can still understand already-constructed SQL queries. Farther, we believe that, given a good example query, researchers can adapt said query to better suit their purposes. This belief lightens the requirements on our system; it does not have to read the researcher's mind, but instead simply generate queries which utilize different SQL constructions and present them in a clear fashion.

3 Query Generation Algorithm

3.1 The Problem

Put simply, the core problem is how to define the utility of queries and select a useful set of queries to offer as examples to the end user. Complicating matters is the fact that we desire to handle cases where little to no external information is available. Generally speaking, no query log will be available if a researcher has just loaded his data into SQLShare, nor will the schema of such uploaded data always be well-defined. Essentially, therefore, the inputs consist of the set of table schema and the data in the tables and little else, while the desired output is a set of about 20 example queries.

Clearly, the best such queries are the ones that need no modification, as they already accomplish what the researcher wants. Just as clearly, in the absence of mind-reading, this standard is impossible to reach without extensive external information about the researcher. However, it is worth noting that example queries are not under significant pressure to be the absolute best examples possible. As long as the provided queries are structurally applicable, some reliance can be placed on the capability of researchers to modify supplied queries to suit their purposes, as discussed in section 2.

Query utility is difficult to objectively quantify - what is “useful” is going to depend on the user’s knowledge and what they want to do. For example, if the user is not very familiar with the schema of tables in a database, example queries which demonstrate good schema coverage might be useful. However, there is a tradeoff between conciseness and schema coverage which becomes especially acute for large databases with sprawling schema - it is not possible to generate 20 short, easily understandable queries which cover most of the schema of a database with several 400+ field tables (see the SDSS[1] database).

3.2 Related Work

There has certainly already been some work done in the area of automatically suggesting SQL to researchers who may or may not be very conversant in it. One which is intimately related with the approach presented in this paper is the work done by Bill Howe et al.[2] which tackles pretty much the same problem as that outlined above, namely, generating example queries without query logs or other external information. However, their approach focuses on generating join and selection clauses while this paper focuses on a different area of query generation, projection. I will discuss this difference of focus more fully in the next section.

Typically, however, approaches to automatic suggestion rely on data which, given the statement of the problem in section 3.1, cannot be assumed to be available. It is not the case that a log of queries is available[3], nor is it the case that user history and/or preferences are available[5]. In addition, the schema may be missing information or simply not well-defined, meaning our methods must work reasonably well in the absence of such information[4].

3.3 Our Approach

In order to simplify matters at this early stage in the research, we restrict the class of queries generated by our system to those which contain only selections, projections, and joins, and do not consider nested queries. Such queries (hereafter abbreviated as "SPJ" queries) have the simplest form and yet retain much expressive power.

SPJ queries split naturally into three components, selection, projection, and joins. Our query generation algorithm, outlined in Figure 1, first finds the most "joinable" attributes (more on this later), then determines the most interesting subset of attributes to filter on from the attributes of the joined tables. Once this has been done, the projection phase attempts to determine interesting attributes available from the join, given the filtration which has already occurred. Once all three stages complete, a complete SPJ query can be produced.

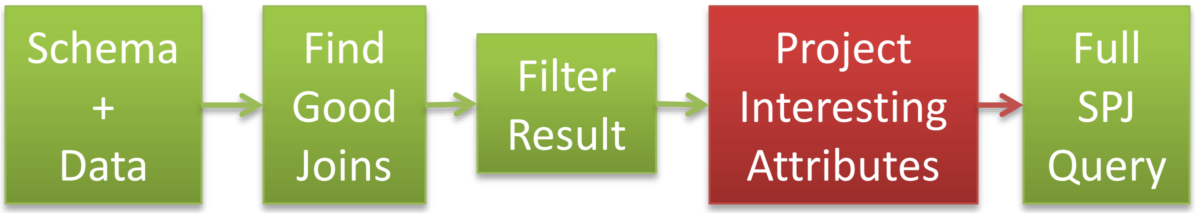

Figure 1

This paper focuses on the stage highlighted in red in Fig.1, the projection, as this is the phase which has had the least research already done on it. Bill Howe and his team at the University of Washington have done interesting initial research on how to determine good attributes to join on between tables, and they are also currently studying the selection phase, determining what attributes would be useful to filter on [2]. Therefore, I determined that the most useful contribution to be made at this point was to research the projection phase.

4 Projection

Projection is an important component of query generation due to the fact that it controls the number of columns in the results from a query. The overarching goal of constructing example queries must be kept in mind – there is no need to project an excessive number of attributes, even if the schema is large. Not only will this clutter the query, detracting from its purpose as an example, it is simply not necessary – we can let the researcher add any other desired attributes or remove unnecessary ones. My goal in this phase is to minimize the amount of modification needed by attempting to automatically ascertain interesting and/or important attributes, while keeping the number of attributes projected to a reasonable number.

To this end, I have hypothesized a set of heuristics for estimating which attributes in a table are most important, which I present here in order of their estimated importance.

Heuristic 1:

The first few attributes in a table schema are generally the most important.

Heuristic 2:

Attributes representing data with more variation are more important.

Heuristic 2.5:

Attributes which appear in a selection equality predicate are not likely to be very interesting.

Heuristic 3:

Attributes with very similar names are often projected all at the same time.

Heuristic 4:

No more than one attribute of a set of attributes representing record-wise identical (or nearly record-wise identical) data should be projected.

The intuition behind the first heuristic stems from my belief that people tend to think of the most important attributes of an explicit or implicit schema first. I use the term "implicit schema" to refer to the organization of data not in a database (for example, in an Excel spreadsheet, or even a raw data file, there is generally a first column, a second, and so on that each represent some aspect of the data). Further, I believe that people whose first language is left-to-right tend to put what they think is more important to the left. Analysis of my own habits when designing schema reveals this trait as well.

The second heuristic was developed from an intuition that more varied data is more interesting. High variation often indicates a high potential for unexpected values, specifically outliers, which are often the chief points of interest in a dataset. Thus, this heuristic biases attribute selection toward picking attributes which display more outliers.

Heuristic 2.5 (so named due to its close relation to heuristic 2) is based on the intuition that if a query filters on an attribute based on its equality to some value, it's likely that the user already knows the value that attribute will take on, especially in the absence of subqueries. For example, given a simple schema (*employeeID, name, city*) suppose I construct a query to select employees which work in the city of Springfield. Such a query would contain a WHERE clause containing the following constraint: … *city* = *"Springfield"* … In this case, projecting the *city* field is unlikely to be interesting, as everyone in the resulting set of records works at the same city, which is known.

After examining some of the sample SDSS[CITE ME] queries, I saw that one query in particular projected psfMag_u, psfMag_g, psfMag_r, psfMag_i, and psfMag_z (among other things). While I was ignorant of the actual purpose of these variables, I noted that u, g, r, i, z and prefixed versions apparently represented related data, since if more than one of them was projected, typically all of them were. Heuristic 3 heuristic codifies the observation that similarity in names often implies similarity in purpose, and that it may well be helpful to project multiple attributes which are closely related to each other. In the example I discussed above, I discovered eventually that u, g, r, i, and z (and their prefixed versions) track data relating to the color of objects, and so they were in truth related.

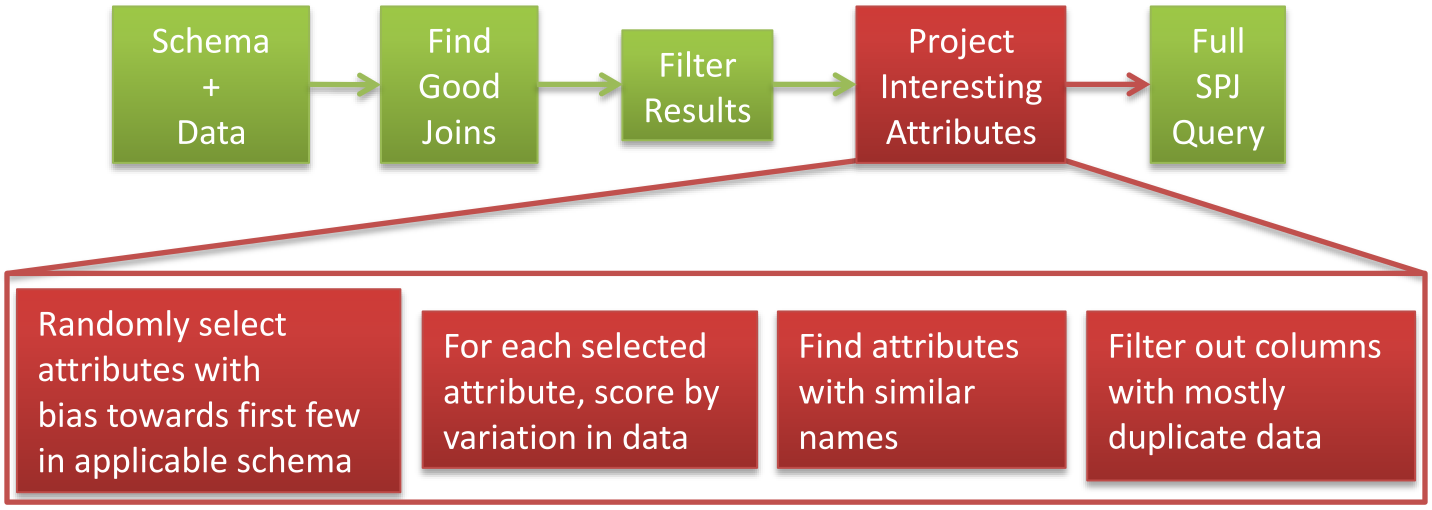
Schema + Data
Find Good Joins
Filter Results
Project Interesting Attributes
Full SPJ Query
Randomly select attributes with bias towards first few in applicable schema
For each selected attribute, score by variation in data
Find attributes with similar names
Filter out columns with mostly duplicate data

Figure 2

Finally, the fourth heuristic is to remove some potential redundancy in tables or views with large schema. It probably isn't worth including data that means the same thing more than once, which is what this heuristic is trying to accomplish. Note that there are subtleties when applying this heuristic - for example, comparing the range and distribution of data represented by a pair of attributes might indicate interesting similarity not redundancy. A more useful attribute comparison for this heuristic, then, might be to compare fields in the same record, record by record - if the fields have little difference on this basis, then it's likely that the attributes in the schema represent nearly identical concepts. This type of comparison is what is indicated by the "record-wise identical" wording in the definition of the heuristic.

These heuristics provide the basis of a methodology to filter attributes available for projection. The method I envision is illustrated in Figure 2, which is an expansion of Figure 1.

5 Evaluation

Evaluation of the heuristics with an eye toward the main goal of generating queries which are useful to researchers is complicated by two factors. The first difficulty is that without the capability to do in-depth user studies, objective evaluation of the success of the algorithm is basically impossible. The closest approximation would be to compare the resulting queries to human-generated example queries, e.g. the example queries provided with the SDSS dataset. The second difficulty is that not all components of the algorithm are complete, which makes the production of queries difficult.

Therefore, in order to provide some evaluation of the utility of my heuristics, I examine how well they apply to an existing set of example queries, the set of queries provided with the SDSS dataset. There are about 60 such queries, which display a great range of complexity.

In order to evaluate heuristic 1 on the query set, I examined the SELECT clauses of 20 queries and determined the index of each attribute in the relevant table schema. I then used this index to compute the following metric:

$$WPS = \frac{attribute\ index}{schema\ size}$$

where WPS is an abbreviation for "weighted positional score." The resulting histogram is shown in Figure 3. Note that the bins of the histogram are ranges of values of the WPS, and the frequency in each bin is the number of attributes from the subset of 20 queries which have WPS scores within that bin's range. The displayed bimodality of this distribution of values is very interesting and was not initially expected – one possible explanation is that interesting attributes may be added to a schema after it has been used in a database as said attributes are revealed to be potentially useful.

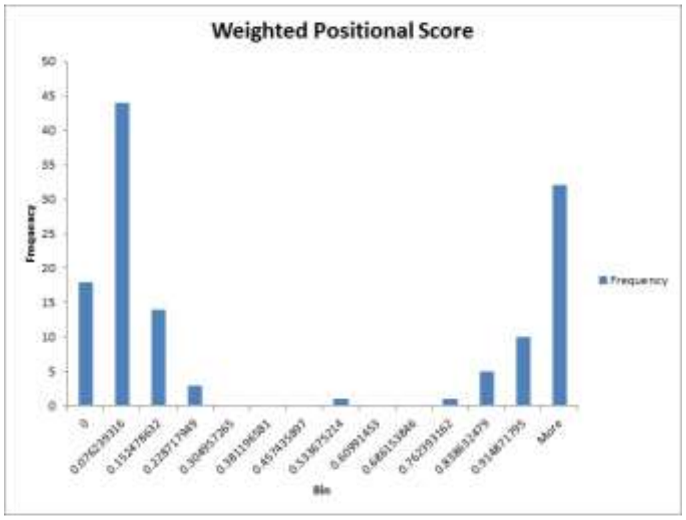

Figure 3

Heuristic 2 proved difficult to evaluate on the SDSS dataset – I chose to measure variation by calculating the ratio of unique values to total values in each column for a set of about 30 attributes. In hindsight, it would not have been significantly more difficult to compute the variation in a more statistical sense, but time did not permit me to refine the measurement in such a fashion. As it happened, even this crude approximation of variation gave interesting results – see Figure 4. As with heuristic 4, an interesting

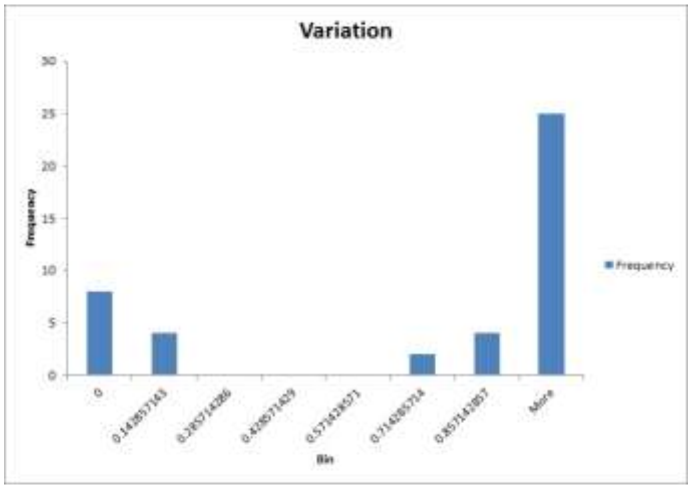

Figure 4

and not entirely expected bimodality appears. One possible explanation for the left hump in the histogram is that it is due to attributes which did not match my approximate notion of variation, but rather the more sophisticated statistical notion, and therefore, while there were many duplicate values, there were also several outliers. It's also possible that the attributes were projected due to some knowledge of their meaning which rendered them useful while still not highly varied.

6 Conclusion

Ultimately, the measurements presented in section 5 are crude and should probably be taken not as proof of the viability of those heuristics but rather strong indications of viability. The bimodal nature of these histograms reveal interesting potential refinements to the heuristics – perhaps heuristic 1 could be modified to state the first *or last* few attributes of a schema are the most important.

To this end, more rigorous measurements are doubtless needed, and over a more diverse set of example queries, not just the ones from SDSS. Unfortunately, it is rather difficult, though not impossible, to acquire such sets of example queries generated by SQL experts.

I feel it important to reiterate once more that it is not the case that the *best* attributes must be selected for projections, selections, or joins. The goal of the algorithm outlined in this paper is to generate some useful example queries, not necessarily find the absolute best example queries.

The area in which I feel the algorithm is currently most lacking is in the production of more different SQL constructions. SPJ queries are simple and easy to understand, but it is also important to show the researcher some more complex queries, or perhaps they will not use the more complex tools simply because they do not know such tools exist.

Another direction in which interesting research could be done is to supplement the example queries with annotations describing the purpose of each clause introduced by the queries, although this is more of an interface design question than anything else.

To wrap up, this paper shows another step on the path toward an algorithm to generate clear and concise sample queries from datasets without already existing usage for researchers with little to no knowledge of SQL.

References

[1] Sloan Digital Sky Survey DR7 (SDSS). http://cas.sdss.org/dr7/en/

[2] B. Howe, G. Cole, N. Khoussainova, L. Battle. Automatic Example Queries for Ad Hoc Databases. 2011.

[3] N. Khoussainova, YongChul Kwon, M. Balazinska, and D. Suciu. SnipSuggest: Context-Aware Autocompletion for SQL. 2011.

[4] X. Yang, C. Procopiuc, D. Srivastava. Summarizing Relational Databases. *Proc. VLDB Endowment*, 2(1):634£645, 2009.

[5] J. Akbarnejad, G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, D. On, N. Polyzotis, J. S. V. Varma. SQL QueRIE Recommendations. *PVLDB*, 3(2):1597–1600, 2010.

[6] SQLShare. http://sqlshare.escience.washington.edu.