Database Exploration and Bellman

Theodore Johnson, Amit Marathe, Tamraparni Dasu
AT&T Labs – Research
{johnsont,marathe,tamr}@research.att.com

Abstract

Large industrial-scale databases tend to be poorly structured, dirty, and very confusing. There are many reasons for this disorder, not the least of which is that the application domains themselves are poorly structured, dirty and confusing. As data analysts, we are often called upon to mine, clean, or otherwise analyze these databases. In this article, we describe the types of problems we have encountered, tools and techniques we have developed to address these problems, and directions for future work.

1 Introduction

As analysts at a very large information-intensive corporation, we are often called upon to analyze business data sets. The analysis tasks range from data mining and analysis to assisting subject matter experts in data quality improvement projects. The source data might be a set of extracted views delivered in delimited ASCII format, or ODBC access to a dozen very large and complex databases which participate in a business process.

The most difficult aspect of these projects is to make sense of the available data. Because of the enormous scope and complexity of AT&T's services and offerings, these data sets would be challenging even if the data were clean, well-organized, and well-documented. In general, they are not, and we spend a significant portion of our time extracting metadata from the data set and from subject matter experts.

In this paper, we describe the kinds of problems we encounter in the data sets and the tools and techniques that we have developed to overcome these problems. We conclude with directions for future research.

2 Problems with Data Interpretation

Understanding the structure and relationships in databases with thousands of tables is a difficult task even in the best of conditions. We often encounter two problems which make the job even harder: poor documentation and data quality problems.

Metadata When we get access to a data set for an analysis, we almost always discover that the documentation and metadata is incomplete, inaccurate, or just plain missing. Sometimes the data set is provided as a collection of delimited ASCII files (or perhaps spreadsheet files). In this case the metadata is often just the file names and the field names. These data sets are either kept using an informal database (e.g. a spreadsheet), or are generated

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

as a query against a database. In either case we need to depend on the data provider to write documentation for us — which is unlikely. A special problem occurs when the data is a query result: the query is a crucial piece of metadata. However the query(s) which generated the data are usually not provided (and often the provider is reluctant to reveal the query).

In other cases, we are provided with logins and programmatic access (e.g., ODBC) to the source databases (or we are given a copy of the database backup). A commercial DBMS provides many facilities for recording metadata. In addition to declaring keys and foreign key joins, we can examine which fields are indexed and what domains, views, and stored procedures are defined. There might also be explanatory comments associated with tables and fields. From the declared keys and foreign key joins, most DBMSs have tools which draw charts of the database structure.

Unfortunately, these metadata are usually incomplete. DBAs rarely label tables with comments, and often do not record keys, foreign key joins, and so on. There are several reasons for this lack of documentation. One is that creating documentation is tedious and unrewarded. The DBA knows what the table means (at least, she does when it is created), and why should she train her replacement? Enforcing key and foreign key join properties can create a high overhead, and can also be dangerous. It can be easier to turn off consistency constraints to make a data set load than it is to debug a data problem. Also, we have encountered serious data quality problems which were caused by the unintended consequences of enforcing foreign key join constraints. Another factor is that enterprise databases change frequently (i.e., because the modeled enterprise does), so the recorded metadata quickly becomes out-of-date. This problem is another disincentive to recording metadata — since its likely to be useless or misleading when the DBA needs it, why should she record it?

There is another reason why we usually find the metadata to be lacking. We are called upon to perform new data mining and data quality studies, and therefore are making new uses of the data. Common uses of the data are likely to be well documented, unusual uses aren't.

Even if the conventional metadata were complete, it usually does not provide enough information to interpret the data set. Some issues are:

- In textbooks, field names fully describe the field contents. The "natural join" makes a strong use of this property — since fields with the same name mean the same thing, the join condition between two tables is a test for equality between all fields with the same name. In practice, there are many names for fields with values which describe the same things. Sometimes the heterogeneity is the result of small misspellings, sometimes the field names are completely different. Often, fields with the same name have very different contents (e.g. *Address* in the IP_CIRCUIT table versus *Address* in the CUSTOMER table). Even if their values are drawn from the same domain, the values might refer to very different things (e.g., customer *Name* vs. salesperson *Name*). When dealing with multiple databases, these problems are even worse.

- Databases often show a surprising degree of heterogeneity. We expect this kind of problem when integrating multiple databases, but we encounter it even in a single database. A very large (1000 table or more) database is often run by multiple DBAs and might refer to multiple logical entities. For example, one group of tables might refer to customer accounts while another group of tables refers to invoices. Names, keys, formats, conventions, normalization, and so on might be significantly different between these different areas. Another cause of heterogeneity is the continual change in the enterprise. Although the DBAs might design a schema which perfectly fits the current state of the enterprise, within six months the database will be called upon to support entirely new types of customers and services. These are often shoehorned into existing tables to the greatest degree possible. As a result, a table will often have multiple disjoint sets of join paths, depending on the flavor of the tuple to be joined. We often find fields with very heterogeneous contents (e.g., describing a customer by name or by numeric key).

- The data in these databases is not static, rather it used (or is generated) by some enterprise processes. A proper interpretation of the data requires an understanding of these processes. However processes are often even less documented than the database.

Data Quality Problems Compounding the difficulties cause by missing, incomplete, or ambiguous metadata are data quality problems. We often find that a task which should be simple winds up being quite difficult because of poor data quality. We need to diagnose the extent of the problems, then explore the database to determine workaround procedures. A selection of the kinds of problems that we frequently encounter are:

Missing Data: We often find that important data does not exist in the database. Sometimes relevant fields are sparsely populated or completely missing. For example, we might want to find predictors of high-revenue customers. In this case we are certain to find that the *Revenue* field of the CUSTOMER table is almost completely NULL. If there is a CUSTOMER_REVENUE table, it is certain to have only a few dozen records. Perhaps the table is the result of a query and the DBA uses an inner join where an outer join would be more appropriate.

Incorrect Values: Even if we were to find a well-populated *Revenue* field, it is quite likely to contain incorrect data (or to be calculated in a way different that what we expect).

Poor Join Paths: Many times join paths are of poor quality – the foreign key join constraints are not maintained. So if we have well-populated CUSTOMER and REVENUE tables, its likely that we can join these tables for only a small fraction of the customers.

Duplicates in Key Fields: Finding a key for a table can be surprisingly difficult. Sometimes a key field (or fields) will contain duplicates. This problem occurs for many reasons. Sometimes unique keys get recycled (e.g., telephone numbers) and the table contains both old and current entities. The data might be the result of a query in which a join creates multiple entries for a few records. Perhaps we are trying to write an ad-hoc query and use a serendipitous join path.

In our experience, problems that we at first attribute to "poor quality data" often turn out to be problems in our interpretation. The CUSTOMER_REVENUE might be built only for customers of a particular class. The *Revenue* field might not include rebates or other pricing adjustments. The CUSTOMER table might refer to corporations as a while, while the REVENUE table refers to entities within corporations. To join these tables, we might need to use a table which associates entities to their parent corporations (we might have a few successful matches in the case of small businesses).

3 Tools and Techniques

Since we are naturally lazy, we have developed tools and techniques to make our jobs easier. The two that we discuss here are *database browsing* tools and *data quality metrics*. Although they are quite different, these tools have two things in common: they cut through the complexity of large systems, and they extract missing metadata.

3.1 Database Browsing

Finding relevant data is difficult in very large (1000+ tables) complex databases. This task is even more difficult when the data is dispersed among a dozen such databases. We found that we need advanced database browsing tools which could help us understand what the database contents and structure are. For example, we usually find "surprises" in the database, and we need to explore the database to determine if the surprise is a data quality problem or a problem in our understanding of the database structure.

The tool we developed, Bellman, *profiles* the database [5] to develop compact statistical summaries of the database contents. When a user browses a database, Bellman uses these summaries to provide informative displays and powerful exploration queries in an interactive fashion.

Many of the facilities that Bellman provides are fairly simple, but surprisingly useful. Bellman collects and interactively displays the number of rows in each table, and for each field the number of unique and null values. This simple function immediately tells about missing data. Bellman also collects information about multiple databases, and can manage multiple open connections to them. While Bellman does not yet support federated queries, the ability to browse and query multiple databases simultaneously is very useful. Other simple but useful tools are a collection of canned queries for sampling, summarizing, and visualizing a table's contents.

Some tools use more sophisticated algorithms, developed by others. Bellman will find all minimal approximate keys of up to four fields on all tables using an algorithm based in part on Tane [7]. We also collect the most frequent values of a field ("heavy hitters") using a sketch-based algorithm [1]. If a table has say 100 fields, we compute the most common field values using a one pass algorithm instead of 100 group-by queries. Most frequent value information is very useful because it gives a feel for the contents of a field, and also because default NULL values are usually the most frequent values (often there is more than one default NULL value). We are currently experimenting with using recently developed approximate string matching algorithms [6] to implement an approximate join engine.

We developed a crucial technology for Bellman, as is described in [3]. For every field, we can interactively compute the *resemblance* of the set of field values (resemblance is the size of the set intersection divided by the size of the set union). We use the resemblance find join paths, and to find data which is "similar" to a sample. With augmented information, we can obtain good estimates of join sizes and whether the join is one-to-one, one-to-many, or many-to-many. Many times two fields will match only after a small transformation. To help in the search for these pairs, we also provide tools for interactive substring similarity queries.

A typical use of Bellman would be as follows. We are asked to make a study of some particular business process. We obtain logins and ODBC access to the relevant databases, then use Bellman to collect database profiles. By talking to Subject Matter Experts (SMEs), we get an understanding of the business process and what our data targets are. Typically, we run into various problems — data is missing, join paths don't work, data mining analyses yield suspicious results, and so on. We begin a database exploration process to find solutions to these problems. We might want to find all data related to frame relay circuits, in which case we will search for frame relay circuit IDs by starting with a sample and using Bellman's resemblance and substring similarity tools. When we find these fields, we use Bellman to explore these tables to determine whether the information they contain is relevant to our needs. We might want to find alternative join paths, either because the documented join paths are broken or because we want a check on suspicious data (i.e., we want correlating information). We use the approximate keys and field resemblance facilities to find the alternative join paths, and the general browsing facilities to evaluate the results.

3.2 Data Quality Metrics

Data quality metrics are intended to assure the data user of the confidence he or she can place in the data. In general, data quality metrics measure the reliability and usability of the data. However, they can also serve to document critical metadata as well. The brief discussion here is taken from a recent book [2].

Conventionally, metrics have been designed for *static* views of the data, with time related aspects incorporated only tangentially, e.g., timeliness of the data. These metrics emphasize rigid constraints such as completeness, accuracy and consistency. In reality, such constraints are often not enforceable. Verifying accuracy would entail expensive inventory checking. Proxies such as sampling or tracking trouble tickets might be insufficient with a potential for built-in bias. Consistency and uniqueness might be impossible to ensure. For instance, a "customer" could mean a billing entity, a corporation, a contract of services , or a piece of equipment. Uniqueness is a stumbling block if different systems require representing an entity with slight variations e.g., "New

Vernon, NJ and “New Vernon NJ. This is especially true of complex legacy systems, which have all kinds of quirks hard coded into the system.

Modern uses and data applications require a much more *dynamic* definition of metrics to reflect accurate conformance to process flows. For instance, there has been a recent focus on auditing databases that support business operation for data quality issues. Such an audit often reveals flaws in the process, in addition to any glitches the data. Based on this, we can define two categories of data quality metrics, — *operational* and *diagnostic*. Operational metrics identify unexpected divergence of process from specification, providing guidance for fixing flaws that manifest as data quality issues. Diagnostic metrics on the other hand, merely identify the flaws in the data and do not provide insight into how to fix them. Completeness is a good example. “The data set has ten percent missing data” is informative but not useful for finding the problem that is causing the data quality issue.

As mentioned earlier, the lack of metadata is a major issue in a data analysis project. Here, we make a distinction between access and interpretation. *Accessibility* can be measured by seemingly bureaucratic criteria such as number of calls, number of people contacted, level of escalation and total time taken from request for access to data to actual physical access to data. *Interpretability* is more nuanced and harder to measure. The schema might be accurate, but there might be hidden rules that are not documented. For example, the people who created the data might have informally compressed the data with rules such as “if the resources allocated to a customer have consecutive serial numbers, record the first and last serial number and set attribute Flag to Y”. We cannot interpret the data if we do not know the rule. Another example in this context is that of censoring in duration data. A call beyond 100 minutes might be censored, so that all we know is that it lasted beyond 100, we cannot distinguish between calls of 101 minutes and 10001 minutes. Such rules are seldom documented and are revealed only by browsing the data and validating the findings with SMEs. Hidden rules show up as glitches in analysis (unexpected spikes in histograms) and data integration (only 10% of the records matched) . The number of analytical passes needed to get reasonable results is a good metric to measure interpretability. Similarly in the operations setting, the number of manual interventions needed (measures automation) and the proportion of data that flows through the process successfully (end-to-end completion) are good metrics. Automation is particularly important since many data quality glitches arise from manual intervention and workaround patches to modify a process designed to automatically complete a multi-step task or transaction. A longer discussion of data quality metrics can be found in [2].

Implementing data quality metrics in a data quality program involves three major stages: (a) data preparation (gathering, storing, integration, manipulation etc.), (b) gathering domain knowledge through interaction with SMEs, e.g., build business rules, devise appropriate metrics and (c) validating the data against the rules and quantifying data quality using the metrics. The three stages are iterated until the experts and the users sign off on the usability of the data. However, the second stage is surprisingly challenging. Subject matter experts are hard to identify (scattered across the company, personnel and project transitions) and often have no incentive to share their time or knowledge. Even if they cooperate, different experts frequently disagree on definitions and details. Finally, there is often a disconnect between the articulation of the rules by the experts and the implementation by the technical staff. A potential solution is offered in [4], where rule based programs are used to encode the expert’s knowledge in a form that can be easily understood and verified by the expert. Such an approach also allows for representing knowledge that is acquired piecemeal, out of sequence and that needs to be updated frequently.

Despite the challenges of implementing data quality metrics as a part of a data quality or data auditing program, there are significant benefits. First, the process of defining data quality metrics results in the creation of metadata and documentation about the data and the process. Second, it puts in place a mechanism for conducting frequent end-to-end audits to catch data quality issues as soon as they arise before they get entrenched and corrupt large sections of the data. Finally, it increases the confidence in the data by putting in place objective, clearly defined metrics that are transparent and cannot be manipulated easily for political ends.

4 Conclusions

The tools that we have developed are very useful in our analysis tasks, letting us quickly resolve seemingly difficult problems. We have found that some of the simplest tools are often the most effective. However, we feel that additional, more sophisticated tools are needed:

- Database structure mining, for example to find clusters of inter-related tables.
- Incorporating dynamic information, such as that extracted from query logs. However, we have found that DBAs are (often justifiably) reluctant to share these logs.
- Integrating SME metadata with extracted metadata.

The digital revolution has only started, the volume and complexity of enterprise information will continue to increase. Software engineers have faced the problem of developing very large scale interoperating code, and in response have developed many design, analysis, and reverse engineering tools and techniques. Analogously, database engineers need similar tool suites.

References

[1] G. Carmode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. In *Proc. Principles of Database Systems*, pages 296–306, 2003.

[2] T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. Wiley, 2003.

[3] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *Proc. ACM SIGMOD Conf.*, pages 240–251, 2002.

[4] T. Dasu, G. Vesonder, and J. Wright. Data quality through knowledge engineering. In *Prof. Conf. Knowledge Discovery and Data Mining*, pages 705–710, 2003.

[5] Evoke Software. Data profiling and mapping, the essential first step in data migration and integration projects. http://www.evokesoftware.com/pdf/wtpprDPM.pdf, 2000.

[6] L. Gravano, P. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an rdbms for web data integration. In *Proc. Intl. world Wide Web Conf.*, pages 90–101, 2003.

[7] Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. Efficient discovery of functional dependencies and approximate dependencies using partitions. In *Proc. IEEE Intl. Conf. on Data Engineering*, pages 392–401, 1998.