Database Exploration Using Database Dynamics

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

Abstract

Data analysts often work at a "distance" from the databases they analyze. DBAs of production databases generally do not give logins to data analysts, who might run large disabling queries. Since large and complex databases are often poorly documented, the analyst faces significant problems in understanding the process of how a database is updated, or even whether the processes are correctly implemented. The current trend of outsourcing compounds the problem, as the outsourcing provider has a strong incentive to reveal as little as possible about their operations. In this paper, we explore the possibility of data mining on database dynamics – that is, performing data mining to explore how a databases changes over time. Two uses for the results are data quality monitoring to verify that the database is being properly maintained, and database reverse engineering to give the analyst additional insight into the structure of a very large but poorly documented database. Our methods use summaries of the database which are generated by our database profiling system, Bellman. This approach allows a user with limited computational resources to mine database dynamics.

1 Introduction

Database users frequently work at a distance from their data sources, and do not know how the large and complex databases they use are maintained and updated. In a large enterprise (commercial, scientific, government, or otherwise), the data producers are often in a different organization than the data consumers and analysts. Database management issues increase the separation between users and their database. The DBAs of production databases generally do not give logins to data analysts, out of concern that the analyst might run a crippling query which interferes with ongoing operations. The current trend of outsourcing compounds the problem, as the outsourcing provider has a strong incentive to reveal as little as possible about their operations. As a result, the data user lacks a crucial piece of metadata: by what process is the database updated? Even if the analyst knows what the processes are supposed to be, she doesnt know whether these processes are being followed. In the case of outsourced databases, no-one in the enterprise may know whether or not the outsourcing provider is maintaining critical information correctly.

Large and complex enterprises are usually supported by correspondingly large and complex databases. The most difficult and time-consuming part of a data mining study is often the identification, acquisition, and cleaning of the data to be mined. Similarly, a complex and time consuming first step in database integration is to

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

understand the databases to be integrated and how they relate to each other. The basic problem is one of complexity. Production databases which support large and complex enterprises have hundreds to thousands of tables; a large and complex enterprise will have dozens of such databases. These databases are often poorly and incompletely documented, and existing documentation might be out of date. Even developers who are familiar with a database can encounter problems while adding functionality to a database to support new applications due to the complexity of the existing database (the disorder and complexity in a database is often the result of the continuing need to adapt an existing database to new requirements). As a result, the DBA of a large, complex database might not be fully aware of the database update process, or whether the process is working correctly. A variety of work has been performed with the goal or side-effect of enabling database exploration and automatic extraction of metadata. For example, database profiling has been studied in the literature [10] and is the basis for some commercial database exploration systems [1, 34]. Data integration systems often include facilities for exploring data integration query results (e.g., Potters Wheel [32], and Clio [18]). Additional relevant research is discussed in the related works section.

In this paper, we propose to aid in the exploration of a database by providing information about database dynamics – that is, how the database changes over time. Our methods for mining database dynamics use the kind of database summary information generated by database profiling systems. By mining database changes over database summaries instead of database snapshots, we can inexpensively perform large-scale analyses of database changes. An enterprise data warehouse might gather copies of dozens of production databases together for correlation, analysis, and data mining. End-users often do not have the resources to host multiple copies of these snapshots; and in any case many organizations have rules which prohibit the proliferation of database copies (for version management and data security reasons). Since the database summaries are small, an end-user can inexpensively support time series analyses of database changes even in very large data warehouses. Further, end-users can store the summaries while they might be prohibited from storing database copies.

There are many applications of database dynamics information; we roughly group them into two types. In database data quality monitoring, we try to detect anomalous or suspicious behaviors while in reverse engineering, we try to understand properties of the database.

Database change information is useful for data quality monitoring. We can place a number of a-priori constraints on the database, then watch for violations. Have important tables or fields been dropped? Did the unfilled orders table fail to get updated? Conversely, was there a change in a table which should be static? Are new values infiltrating a field which should have a limited domain? We can also detect possible data quality problems by searching for deviations from past behavior. For example, a previously static table or field which exhibits a significant change can indicate either a data glitch or a change in procedures. An active entity which fails to change can indicate a data warehouse loading problem or a problem in production processes.

One common example of a reverse engineering activity is to try to find the important tables in the database. This task is surprisingly difficult, e.g. in a poorly documented database with hundreds or thousands of tables. The database explorer can use a variety of clues to find interesting tables: table and field names, documentation, the number of rows in a table, and samples of row and field values. Several database exploration systems provide this kind of information [1, 10, 34], but there is still a missing piece: the update activity on the table. Is the table regularly updated or is it static? If the table is regularly updated, what is the nature of the update – small or large; added and deleted rows, or changes in field values? Which fields of a table are being updated? These properties indicate how the table is used. For example, a static table might be a relic that is no longer in use. A table of unfulfilled orders should exhibit large changes from week to week, while a table of customers should exhibit relatively small changes.

In this paper, we explore the use of time series of database profiles, generated by our Bellman profiling tool, to mine information about database dynamics, which to our knowledge has not been done before (except for some degree in [23]). Our contributions are:

1. Evaluating the effectiveness of different types of profile data for performing database change detection,

2. Developing methods for mining information from profile time series, and

3. Evaluating these methods on profile time series from two production databases used by a large enterprise.

Our analysis techniques are deliberately simple exploratory methods – the contribution of this paper is the extraction and exploration of a novel data set. Nevertheless, these techniques can find many interesting properties of the databases we explore.

1.1 Related Work

The results in this paper use data gathered by a database profiling system [13] small but informative summaries of database contents. Commercial providers of profiling systems include Similarity Systems [34] and Ascential [1]. For our research we use data collected by Bellman [10] developed and used at AT&T. These summaries can include keys and functional dependencies [6, 20, 30, 33]. In [1] the authors present techniques for creating summaries based on information-theoretic clustering. However, Bellman does not incorporate this kind of profile data so we do not explore its use for change detection in this paper.

Database reverse-engineering is a pre-requisite for schema mapping and database integration. Several schema mapping efforts have incorporated database exploration tools [12, 17, 18, 29, 30]. Some schema mapping tools also allow the exploration of the database integration results [18, 32].

Recent work has developed tools for keyword search in a database [1, 3, 5, 21, 22]. Other types of advanced search tools include regular expression indexing [8, 19]. A related topic is that of approximate join for which we refer the interested reader to the tutorial by Koudas and Srivastava [16].

Some of the results in this paper can be considered to be data mining for data quality. Other methods include the use of multiple independent models [27], non-parametric outlier identification [26], and automated learning to detect data quality problems [25].

Most commercial DBMSs optionally allow the logging of all queries submitted to the system. An analysis of the query log could provide a more direct method for reverse-engineering a DBMS [28]. However, query logs are usually not available to end-users, and their contents can be difficult to interpret.

Some recent work has addressed detecting changes in data sets [14, 15, 24]. These papers treat the contents of a table (or stream) as a multivariate distribution and look for changes in the distribution (in [14] by measuring distance through classification models, in [15] by partition-wise comparison, while [24] tests stationarity by measuring distance between distributions). In [7], the authors use a sampling method to detect the frequency by which a data source (web pages or data warehouses) changes. The focus of this paper is to minimize the cost keeping a local copy of the data up-to-date. A side effect of the algorithm presented in [7] is to estimate the rate at which data sources change, but requires a copy of the data set. The most closely related work is [23], which monitors the rate of change in text databases by measuring longitudinal change in a summary (an inverted list index). However, this study only measured table changes, used a single summary, and did not attempt to infer properties of the table (the issue was finding a schedule for loading a local copy). In contrast to previous work, we present a method for mining database dynamics which extracts information about database management and update procedures, and is the first study of its kind that we are aware of.

2 Profile Data

The profile data that we use for our study is that collected by the Bellman profiler. The basic profile data collected by Bellman is:

- The schema of the database.
- The number of rows in each table.

- The number of distinct values of each field.
- The number of NULL values of each field.
- The frequency distribution of each field.
- The top ten most common values of each field, and each values frequency of occurrence.

Basic profile data is collected by the commercial database profiling systems that we are aware of [1, 34]. The commercial database profilers generally collect some additional information, for example the keys and functional dependencies in the tables. While Bellman can also collect key and FD data, the longitudinal data set that we had access to did not include them (computing keys and functional dependencies is expensive, so they were not collected). Commercial profiling systems can also collect row samples; Bellman does not. While it is clear that, e.g. keys and functional dependencies are very useful, e.g. for data quality monitoring, we cannot report any results about them.

Bellman also collects minhash signatures and sketches of some database entities. While these approximation techniques are well-discussed elsewhere [10], we provide a brief review here to make the paper self-contained.

The resemblance between two sets is the size of the intersection of the sets divided by the size of their union.

$$\rho = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

The resemblance of two sets can be approximated by a minhash signature: Let hi be a hash function defined over the domain of set A. Let $s_i(A) = \min\{h_i(a) | a \in A\}$. Then $\Pr[s_i(A) = s_i(B)] = \rho$. We repeat this experiment N times, and the minhash signature of a set A is the collection of these experiments: $S(A) = \{s_1(A), \ldots, S_N(A)\}$. Bellman collects minhash signatures over multisets (e.g., the multiset of values of a field), providing an opportunity to collect an approximation to the frequency distribution of the multiset: $M(A) = \{m_1(A), \ldots, m_N(A)\}$, where $m_i(A)$ is the number of times that $h_i^{-1}(s_i(A))$ occurs in A.

Bellman also makes use of random vector projection sketches. Let V be a d-dimensional vector, and X a d by k dimensional matrix, $k \ll d$. Then the sketch of V is

$$SK(V) = V \cdot X \tag{2}$$

Where SK(V) is a k-dimensional vector. The L_2 distance between two vectors V_1 and V_2 can be approximated by

$$L_2(\hat{V_1, V_2}) = L_2(SK(V_1), SK(V_2)) \tag{3}$$

New fast sketch techniques, e.g. the count-min sketch [9], have been proposed, but they have not been incorporated into Bellman at the time of our study.

The sketch and signature profile data that Bellman collects is

- The minhash signature of value each field.
- The minhash signature of the values of the rows of each table.
- The minhash signature of the set of 3-grams of each field.
- The sketch of the frequency vector of the multiset of 3-grams of each field.

3 Experimental Setup

Bellman is used on a variety of projects within AT&T, providing a source of data for our experiments. One project in particular is a data warehouse consisting of copies of production databases, to be used for intensive data mining analyses. Currently, this data warehouse contains 15 databases updated weekly, with a total of 2899 tables. We selected two of these databases, which we call A and N, to study. We chose these particular two because among the large and complex databases in the warehouse, they were the most reliably updated and the best understood by the local analysts (but the documentation for these databases was quite meager). These databases also represent the distance problem that the analysts face, as corporate policy forbade the analysts from making a local copy (for version management and data security reasons), and the management of these databases had been outsourced.

We arranged with the Bellman developers to collect snapshots of the weekly profiles, timestamped with the profile collection time, for two of the fifteen databases. At the time of this writing, the first database (A) contains 385 tables and 3522 fields, while the second database (N) contains 508 tables and 3547 fields. We collected 33 profile snapshots, allowing for 32 successive weeks of comparison data.

We supported our experiments by making a pre-analysis of the data, computing time series of table and field self-resemblances by value, by q-gram, and by q-gram distance; and also summaries of changes in other field statistics. Databases A and N contain more than 420 million rows and represent about 100 Gbytes of data; computing their profiles requires about 8 hours of computing time (on a 2.8 GHz, 2-processor server) and results in about 500 Mbytes of profile data. Computing the summary tables requires about an additional hour of compute time, after which all queries used to generate the data in this paper ran in real time. Since the data warehouse is profiled weekly anyway, the results in this paper provide database dynamics information almost ”for free”.

4 Experiments

In this section, we explore the efficacy of different types of profile data to detect various types of changes in a database. This section is divided into subsections based on the type of change we are trying to detect.

We do not examine the use of schema information in this paper. Although schema information is quite useful for detecting added or dropped tables or fields, detecting these changes is straightforward (this kind of a data quality alerting facility was added to Bellman without the need for research). We note that we found many examples of added and dropped tables and fields in the course of our study.

This study is limited by the types of databases we study, the types of profile data collected on them, and the local knowledge about these very complex databases (but obtaining the data used for the experiments was a multi-year effort). The results of the experiments indicate the type of information that can be found by mining database change information. Different applications will have different characteristics, so one cannot expect the precise results of this study to be universally applicable. Still, databases A and N show remarkably similar behavior, and provide an indication of what one might find in other databases.

4.1 Database Change Detection

In the context of a data warehouse receiving weekly updates, an important question is, did the database refresh occur successfully? Although it would seem that this is a simple question to answer (e.g., look at the server logs), in a large enterprise there can be complications. The data warehouse we studied is managed on a shoestring budget by an organization remote from its users. The data loads are accomplished using automated scripts, and the DBAs do not have the resources to analyze the loading process. The database load might fail for many reasons: the backup image was not transferred from the production database, resource limitations such as lack of disk space, a failure in an ETL process, and so on. So, the task of evaluating the quality of the database refresh falls to the users.

There are two types of profile data which naturally suggest that they can be used to measure a table change: the change in the number of rows in the table, and the resemblance of the row values of a table to themselves (table self-resemblance). Self-resemblance would seem to be the more sensitive of the two tests, as it can detect changes to a table made using row updates only. However, we approximate the table self-resemblance using minhash signatures, which cannot reliably detect small changes. To evaluate these two methods, we collected 12249 week-to-week comparisons of profile data for the A and N databases. We counted the number of table changes detected using combinations of row count change and a table self-resemblance of less than 1. A summary is in Table 1.

Row count and self resemblance	2256 (18.4%)
Row count only	2857 (23.2%)
Self resemblance only	2520 (20.6%)
Row count or resemblance	3121 (25.5%)

Table 1: Number of table changes detected.

By plotting the number of changes over time, we can track aggregate database behavior. In Figure 1, we plot the week-to-week fraction of the tables in database A with a change detected by self-resemblance, change in row count, and both. Figure 2 shows the same chart for database N.

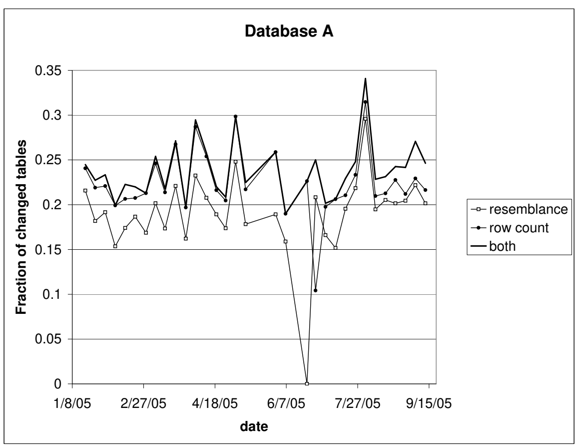


Figure 1 : Tables with a detected change.

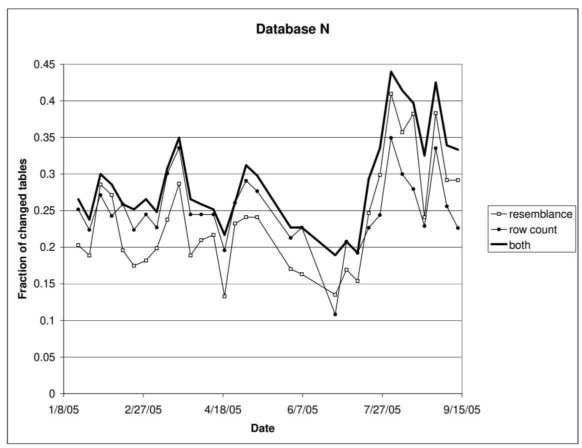


Figure 2 : Tables with a detected change.

One striking feature of these databases is that most of the tables are static – only 20% to 30% of the tables in Database A and 20% to 40% of the tables in Database N change from week to week. We found this property to hold for all of the 15 databases in the data warehouse. Therefore, looking for changing tables can be an e ffective way to reverse engineer a database. We explore this topic further in Section 4.2.

In addition, Figure 1 indicates a data refresh problem in Database A for its 6/21/05 data point. Using self-resemblance indicates no week-to-week change, while using row count does. Further investigation shows that while the average percentage change in the row count (of tables whose row count changed week to week) is 5.9%, the row count change during that week is only .4%. We later confirmed a data refresh problem.

A comparison of the changed tables detected by self-resemblance and row count indicates a subtle but distinct change in database dynamics when we compare pre-6/21 to post-6/21 behavior. The fraction of tables which change increases dramatically in Database N. In addition, self-resemblance accounts for a larger number of detected changed tables. For example, in Database N more changes are detected by self-resemblance than by row count after 6/21, but fewer before 6/21. We confirmed that data loading procedures were improved during this period in response to user complaints.

4.2 Table Change Detection

The results from Section 4.1 suggest that row counts are generally a better indicator than self-resemblance for detecting changed tables. However, in that section we are primarily interested in determining whether or not tables did change. In this section, we will also be interested in the degree by which the table changed. Therefore, we need to determine which measure is more sensitive in detecting the degree of change. In Figure 3, we plot the degree of table change detected using table self-resemblance vs. the degree detected using row counts for the 3121 instances where we detected any table change using either method. Unsurprisingly, self-resemblance is almost always a more sensitive indicator of change (as shown by the fact that almost all data points are below the diagonal).

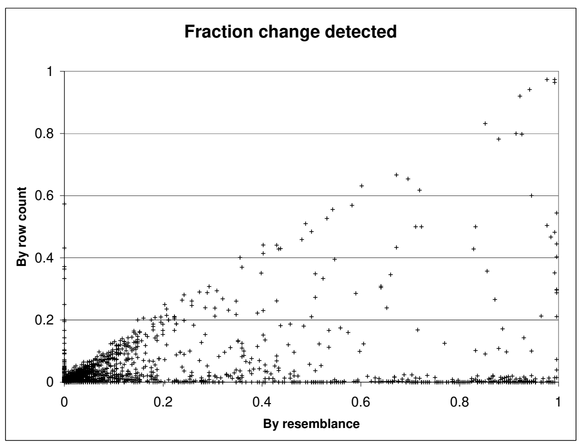


Figure 3 : Fraction of change detected in tables using row count and resemblance.

We can try to classify the type of updates to a table by the correlation of the fraction of change detected by row count to the fraction of change detected by self-resemblance. If the change is primarily due to insertions or deletions, the resemblance change and the row count change should be strongly correlated as indicated by the diagonal line in Figure 3. If the change is due primarily to updates, the row count should stay the same while the resemblance should show a large change – as indicated by the horizontal line in Figure 3. Changes due to a mixture of inserts, deletes, and updates should lie between these two lines.

Next, we can characterize how a table changes: Is it mostly updates, inserts or deletes, or a mixture. Is the mixture consistent? We collected the 2520 instances where table self-resemblance indicates a week-to-week change in a table. Next, we computed the ratio of the fraction change in row count to the fraction change in self-resemblance:

$$ratio = \frac{|count_{old} - count_{new}|}{1 - \rho} \tag{4}$$

We next analyzed this data on a per-table basis, for the set of tables with at least four week-to-week changes (80 tables in database A, 66 tables in database N).

We computed the coefficient of variation (square root of the mean squared error divided by the mean) for the fraction change ratio, and found a wide range of coefficients. To limit our scope to well-behaved time series, we selected those tables with at least four changes detectable by resemblance and a coefficient of variation less than .75 (an arbitrary value that we selected by eyeball). We are left with 46 tables in database A, 60 tables in database N.

In these 106 tables, we found 26 with a ratio of 5% or less, and 27 with a ratio of 85% or greater (Figure 4 shows the distribution of the ratio, by sorting tables on their change ratio). Since we chose these tables because their change ratio time series is reasonably well behaved, we find that we can distinguish between these tables based on the types of updates they normally receive. For example, consider the change ratio time series for the three tables in Figure 5. One table ($ratio < .05$) changes primarily through updates, another ($ratio = .5$) through a mix of inserts, deletes, and/or updates, and the third ($ratio > .85$) primarily through inserts (we verified that its size increases, not decreases, over time).

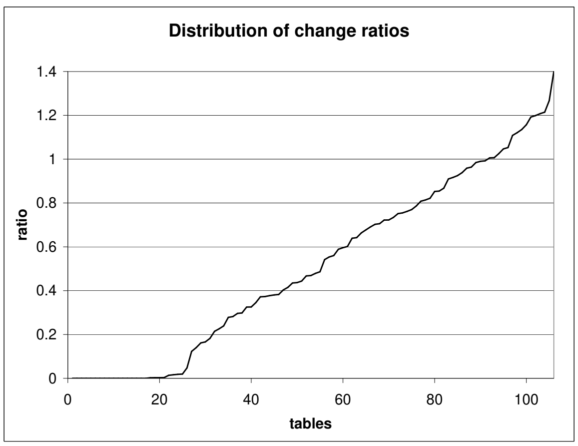


Figure 4 : Row count vs self-resemblance change.

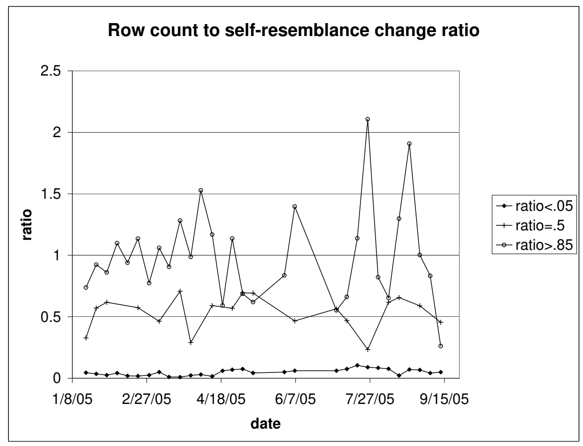


Figure 5 : Change ratios for three tables.

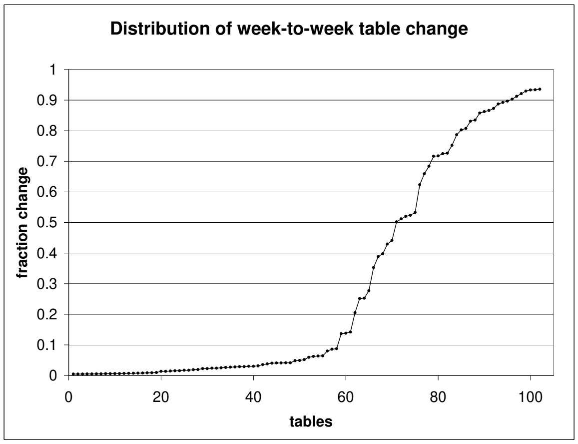
We can also classify tables by the degree to which they change from week to week. We selected the tables which changed at least six times as indicated by self-resemblance, computing the change to be $1 - \rho$. The result is 63 tables in Database A and 60 tables in Database N. Next, we computed the trimmed coefficient of variation for these tables[1] and selected those whose coefficient of variation is .75 or less. The result is 54 tables from Database A and 48 tables from Database N.

These tables show a wide variation in the fraction by which they change week to week, as is shown in Figure 6. 22 of the 102 tables change by an average of 1.5% or less per week, and 13 of the tables change by 85% or more. Figure 7 shows the week-to-week fraction of change for a table with small, medium, and large week-to-week change, respectively (the large-change table was added in July 2005).

Yet another way to classify tables is by whether or not they tend to recycle their rows. We call a table which recycles its rows convergent, while a table which tends to create new rows divergent. We look for convergent tables by comparing the time series of four week resemblance changes to an estimate of the four week resemblance change computed using 1-week resemblance changes. We estimate the 4-week resemblance by the product of the previous four 1-week resemblances, and the 4-week change by one minus the four week resemblance.

We found that this simple estimator generally works well. That is, we found that most tables are divergent. For the 63 tables in Database A and 60 tables in Database N which have at least six changes indicated by resemblance, we computed the ratio of the actual four-week change to the estimated four-week change. For

[1]For this analysis, we ignore zero change ($\rho = 1$) data points, because they might indicate only a failure to refresh.



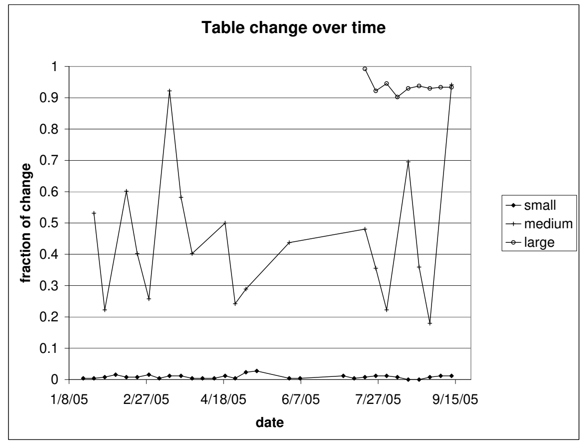
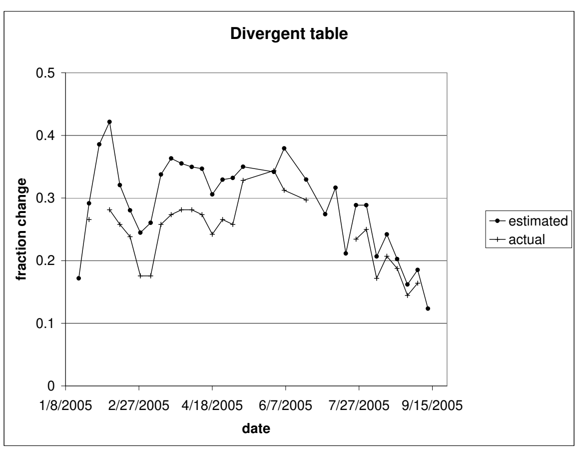


Figure 6 : Fraction of change week-to-week.

Figure 7 : Small, medium, and large week-to-week change.

each table, we counted the number of times that this ratio is .7 or less. We selected the tables with a count of six or higher as the convergent tables. We found 10 convergent tables in database A, and 6 convergent tables in database N.

Figure 8 shows a comparison between the estimated and actual 4-week resemblance changes in a divergent table. This table actually shows a change in its behavior. Before 5/9/2005, the actual change is somewhat less than the estimated change (about 77% of estimated change, above our somewhat arbitrary 70% threshold). On 5/9/2005 and after, the estimated change closely tracks the actual change. Figure 9 shows estimated and actual four-week changes for a convergent table. Its behavior is different than that shown in Figure 9, as actual percentage change is persistently about half the estimated change – indicating that this table tends to recycle its rows.



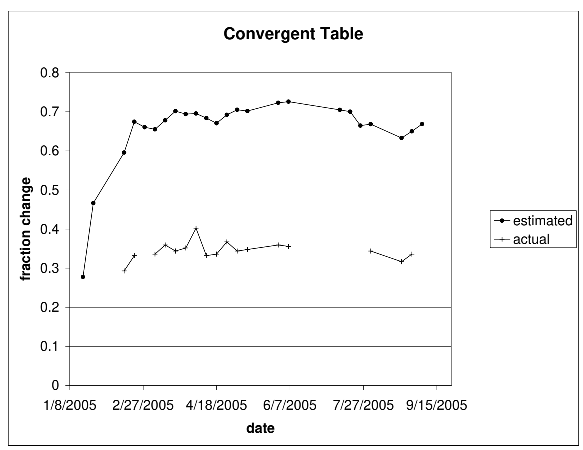


Figure 8 : Divergent table, 4-week changes.

Figure 9 : Convergent table, 4-week changes.

There is one more type of information we can use for classifying changes in tables changes in the q-grams of their rows considered as strings. Recall from Section 2 that we have two types of q-gram information: q-gram resemblance and q-gram distance. To evaluate these metrics, for each of the 123 tables with at least six changes detected by value self-resemblance, we computed the ratio of the week-to-week change detected by value self-resemblance to the change detected by q-gram self-resemblance, and to the change detected by q-gram distance. The result is shown in Figure 10.

Neither q-gram resemblance nor q-gram distance is a good metric for determining that a table has changed. However, the distribution of ratios for q-gram resemblance suggests that this ratio can classify tables based on the rate at which they change their textual content (the peaks in both charts are due to only two tables). The changes detected by q-gram resemblance versus q-gram distance are only moderately correlated: we found an R^2 value of only .33 when performing a linear regression between the two variables, and that only after removing two outliers. Since row q-gram resemblance is more sensitive in detecting textual changes, we recommend it as the better mechanism for classifying this kind of table change.

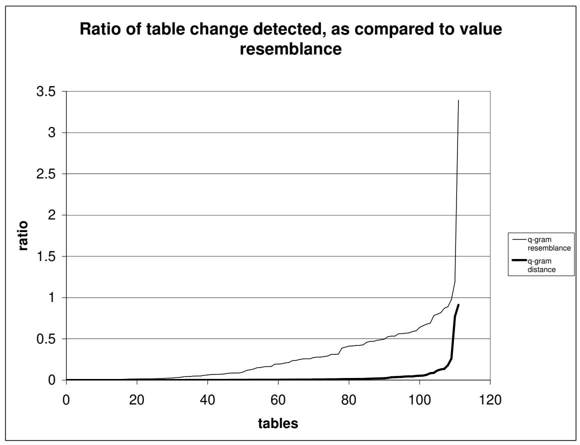


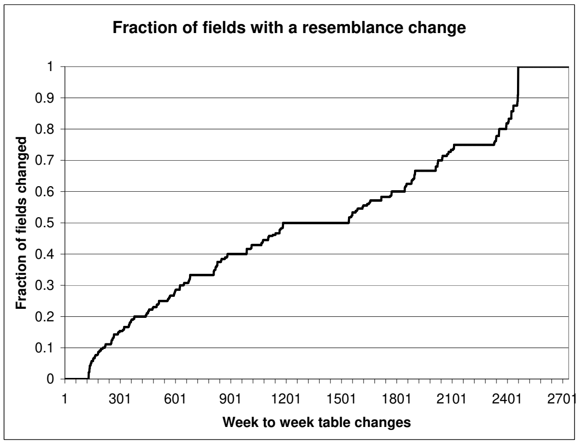
Figure 10 : Row value change versus row q-gram change.

4.3 Field Change Detection

For detecting changes in fields, we have a wealth of information. In addition to the value and q-gram resemblances, and q-gram distance information, we have the count of the number of distinct items, the count of the NULL values, the top-10 values, and an approximation to the frequency distribution of the values of the field. We limit the scope of our experiments to fit within page limits and minimize reader fatigue. We will use value self-resemblance and frequency distribution information to characterize the nature of the updates to the tables, and q-grams, count distinct, count NULL, and top-10 information to discover unusual events.

In Section 4.2, we search for "hot" tables – those that get updated – and found that a small fraction of the tables in the database get updated from week to week. A natural extension is to search for "hot" fields. That is, can we identify a small collection of fields in a table which account for the updates? For every occasion in which a table shows a table self-resemblance change, we compute the fraction of that tables fields which also show a value self-resemblance change. We plot this fraction for the 2732 week-to-week table changes in Figure 11. We find that in a large number of tables, the values in only a small set of fields have value changes on 130 occasions, there are no value changes in any fields. Conversely, on 274 occasions, 90% or more of the tables fields show a value resemblance change.

Next we can ask whether there are fields of a table that are persistently "hot", in terms of value resemblance change. We selected the 123 tables which showed a table self-resemblance change on six or more occasions and analyzed the number of times each of their fields show a change. We only considered those fields which exist in every observation (there was a small but significant number of added and dropped fields). The result



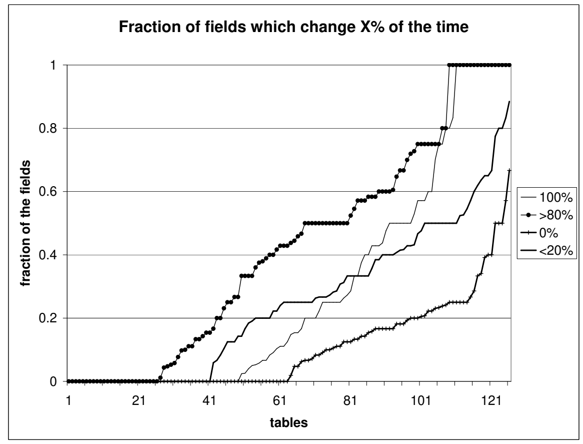


Figure 11 : Fieldswith a value resemblance change.

Figure 12 : Fields with persistent changes.

is in Figure 12. Although there are a significant number of occasions in which none of a tables fields show a resemblance change, there are no tables whose fields persistently never, or even rarely, change. However, for most tables there is a subset of hot (frequent value change) and cold fields (infrequent value change).

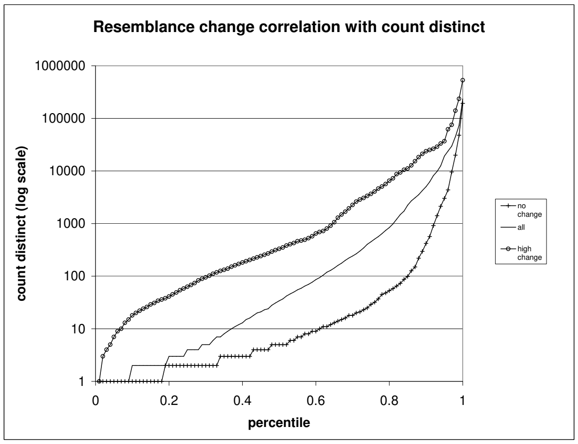
Is there significance to the set of fields whose values are not changing? In most databases there is a large collection of fields with only a few values, e.g. Male/Female. One would expect that values in these fields do not change. To answer this question, we correlated the number of distinct items in a field to its resemblance change during the occasions where the fields table has a week-to-week resemblance change. We plot the distribution all of fields, for those fields which do not have a resemblance change, and for those which have a high resemblance change (larger than or equal to the table resemblance change) in Figure 13. As expected, fields with large changes tend to have significantly more distinct values than usual, and fields which do not change tend to have significantly fewer distinct values than usual. However, there is a significant fraction of fields with few distinct values but a large resemblance change, and also of fields with many distinct values but no resemblance change.

We can use the change in the distribution of a fields values as a more sensitive metric. Of the 34727 field/occasions under consideration, the self-resemblance is 1.0 in 20327 of the fields, but the distribution is identical in only 6555 of these. A failure for a field to change in distribution strongly suggests that its table is changed by updates on other fields. Are there fields which persistently do not change their distribution? We collected 1474 fields which occur six times or more in tables[2] and which show a resemblance change, and counted the number of times when each of these fields showed zero change in resemblance and distribution. We plot the results, in terms of frequency of no change, in Figure 14. We find 90 fields which never change, and 144 fields which do not change 80% of the time or more when their table changes. These non-changing fields occurred in 24 of the 207 tables which showed six or more resemblance changes.

For the remainder of the field change data (q-gram resemblance, q-gram distance, number distinct, number null, top-10), we search for significant changes in the field. We collected field change data such that the fields table showed a week-to-week change and for which we could join field statistics, table row counts, q-gram resemblance, and q-gram distance. The result was 18078 rows.

- We compared the week-to-week change in the number of NULL values of each field, and divided by the (ending) number of rows in the table. We found 59 occurrences in which the number of NULL values changed by 20% or more.

[2]We need this restriction because some fields were added or deleted during the course of the study.



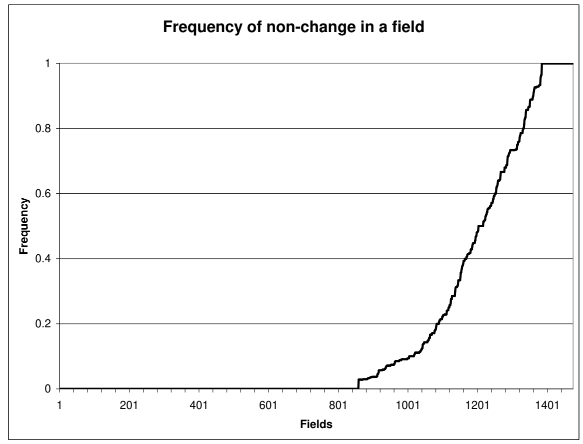


Figure 13 : Distribution of field count distinct values. Figure 14 : Frequency on non-change.

- We compared the week-to-week number of unique values of each field, and divided by the (ending) number of distinct values. We found 407 occasions in which the number of unique values changed by 20% or more.
- We computed the number of top-K values which were common week-to-week and divided by the number of top-K values at the end of the week. We found 1308 occurrences in which 20% or more of the top-K values changed.
- We found 630 occurrences in which the week-to-week q-gram resemblance indicates a change of 20% or more.
- We found 186 occurrences in which the week-to-week q-gram distance is .2 or more.

Unsurprisingly, these indicators of unusual behavior in a field are strongly correlated.

4.4 Time Series Analysis

Since we have a time series of table change data, we can try to look for tables which change in similar ways. If the changes are strikingly similar enough, we might even suspect that there is a similar process which updates the tables. To do this, we need to cluster tables based on their time series. For this experiment, we use the week-to-week table self-resemblances.

A common approach to clustering time series involves summarizing the time series using summaries such as Fourier transforms and clustering the resulting fixed length vectors. However, model based clustering can fail when the assumptions are not true. We know very little about how tables are expected to change from week to week; therefore we use the fast, simple, and nonparametric method to cluster the time series, described in [11]. This method computes simple nonparametric summaries of the time series, and then clusters time series based on their summaries.

In this specific instance, we use the following three descriptors: average resemblance, the number of times the resemblance is less than 1.0, and the smallest resemblance (i.e. the biggest change). We then run a fast k-means clustering algorithm (FASTCLUS from SAS) with 10 and 20 clusters. While seemingly simple, the technique is very fast and effective, resulting in clusters that clearly identify different types of behavior. The

”goodness-of-fit” criteria provided by the software (e.g. R-square of 0.98) all indicate that clustering was effective in explaining the variability of the three clustering attributes. Note that our aim here is to create approximate groupings which explain behavior, rather than exact, optimally separated clusters. The approximate clusters enable us to partition the data into smaller pieces for further examination of the peculiarities of individual time series.

Since Database A and Database N show similar behavior, we concentrate on Database N. We found that 20 clusters produced many singleton clusters (i.e., capturing outliers), so we chose to examine the 10-cluster results. Figure 15 is a bubble chart showing the minimum resemblance and number of detected changes axes for database N (the bubble size indicates the size of the cluster.

We picked a few of the 10 clusters to show as representatives. Figure 16 shows a low-activity cluster, while Figure 17 shows a high activity cluster, demonstrating that clustering technique does tend to group similar behaving time series. In Figure 18, the tables are mostly quiescent, but many of them show a significant change at week 27. These tables all show an anomalous change of similar magnitude at the same time period, suggesting that there is a similar process behind this update.

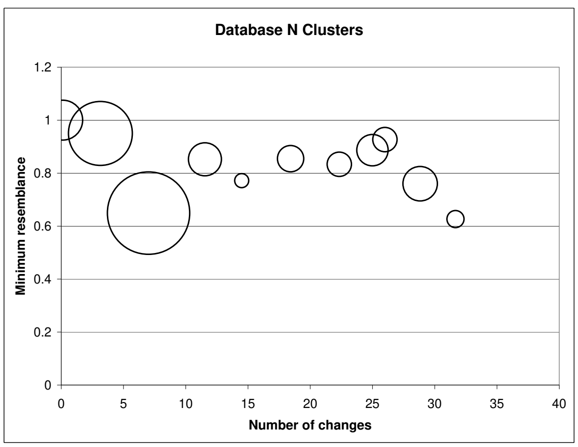


Figure 15 : Clusters in Database N’s value resemblance time series.

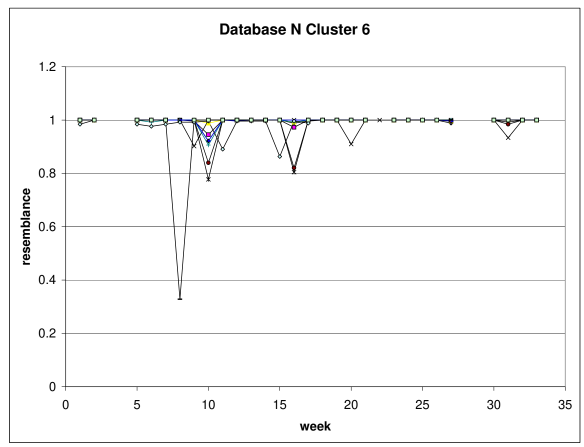
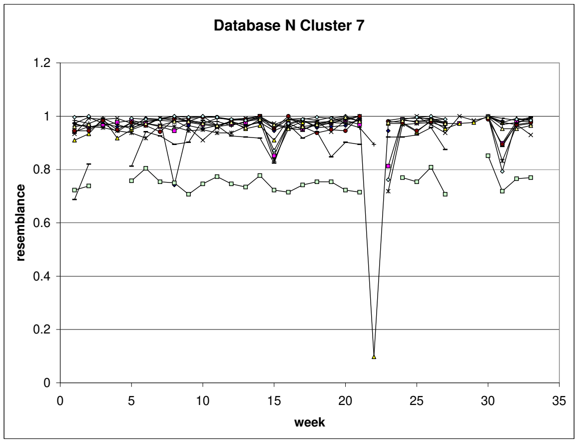


Figure 16 : Low activity cluster.

For our final experiments, we attempt to find anomalous behavior in the week-to-week table self-resemblances using time series analysis. We use the table self-resemblance time series and search for anomalous table updates. At first we tried to capture the structure in the resemblance time series using linear regression models of various sorts. However, the data was not amenable to such models due to high correlations and singularity issues. We then switched to a control chart type of approach, modified slightly to meet our needs.

We computed a set of four-step moving averages for each time series. We then computed the average and standard deviation of these moving averages to serve as a central reference line of the control chart and basis for the confidence or error bounds respectively. We used 2-sigma limits, and flag an alert whenever the self-resemblance is outside these bounds. Analogously, we also computed the median of the moving averages and 10th and 90th quantiles to serves as error bounds. However, we believe that these bounds are less reliable than the mean and standard deviation based ones due to the limited amount of data.

The control chart technique raised at least one alert on almost half of the time series. Many of these alerts indicated an anomalous update to a table; Figure 19 shows an example. There were also many alerts as shown in Figure 20. Here the most extreme alert indicates a failure to update the table.



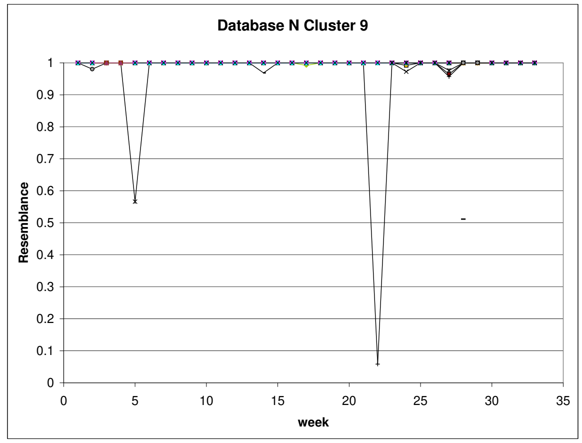


Figure 17 : High activity cluster.

Figure 18 : Cluster indicating an anomolous correlated change in tables.

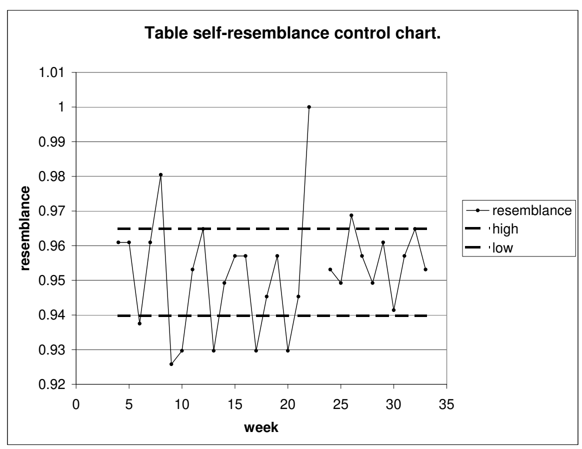


Figure 19 : Anomalous table update.

Figure 20 : Failure to update a table.

5 Conclusions

A common problem faced by database users and maintainers is that of understanding what is in the database. A production database for a complex enterprise is generally a quite complex thing itself, separated from the end-user, and often outsourced. Frequently, even the DBAs of a database dont completely understand its operations, sometimes leading to disastrous mistakes in maintenance and upgrades.

Data warehouse maintainers and users have even fewer resources than the DBAs for understanding what is happening in a database. In fact, this project was motivated by problems encountered in using a large data warehouse used for cross-database analysis and data mining; in particular to determine whether or not weekly database refreshes to the (remotely managed) warehouse were successful.

Recent research has developed new tools for exploring the contents of complex databases. In particular, database profiling has been developed both in research efforts and as commercial offerings. These tools provide a wealth of information about the structure and contents of a snapshot of a database. However, an important dimension of information is missing the database dynamics. That is, what parts of the database are changing, and in what way are they changing.

Data warehouse maintainers and users typically do not have access to query logs of production databases, which can provide direct evidence of database dynamics. Maintaining historic snapshots of database contents in a data warehouse is often not feasible in a large federated warehouse, and analyzing large database snapshots for differences can be prohibitively expensive. A data warehouse maintainer might perform periodic database profiling on the data warehouse contents, to provide metadata to the warehouse users and to screen for data quality problems. Historic database snapshot profiles provide an excellent data source for data mining to discover database dynamics. Being small but informative summaries of database contents, they are easily stored and mined even on inexpensive equipment, and can reveal trends in databases changes.

In this paper, we explore possibilities of mining database snapshot profiles. Because of the novel nature of the data set, our analyses use deliberately simple exploratory techniques. Nevertheless, we are able to extract from our sample databases a significant amount of reverse engineering and data quality information, including:

- Baselines for changes to a database.
- Identification of hot tables.
- Classification of changes to tables by a variety of criteria, including rate of change, insertions vs. updates, and convergent versus divergent change.
- Identification of hot fields within a table.
- Identification of suspicious changes to a field.
- Identification of tables with correlated updates.
- Identification of suspicious updates to a table.

Because of the wealth of information which can be extracted for free from a database profiling system, the Bellman developers are using the results of this study to add new browsing and alerting functionality to the Bellman system.

5.1 Future Work

A great deal of future work is possible; in particular when specific domain knowledge is available. Our particular interest is to make a deeper exploration of the meaning of what we can discover. For example, can we correlate the anomalous events we found in Section 4.3 with particular data quality problems. Conversely, if we identify data quality problems, can we correlate them to events observed in the database change data?

Our access to this kind of domain-specific information was very limited (and formed part of the motivation for this research). We hope that the results of this research will motivate the development of database change detection systems which, when put into production use by subject matter experts, can yield more detailed experimental results.

References

[1] G. Antoshenkov. Byte-aligned data compression. U.S. Patent number 5,363,098.

[1] Ascential. ibm.ascential.com

[2] BANKS: Browsing and Keyword Searching in Relational Databases. B Aditya et al. VLDB, 2002.

[3] DBXplorer: A System for Keyword-Based Search over Relational Databases. S. Agrawal, S. Chaudhuri, and G. Das. ICDE 2002. pg 5-16.

[4] Information-Theoretic tools for mining database structure from large data sets. P Andritsos, RJ Miller, and P Tsaparas, SIGMOD 2004.

[5] ObjectRank: Authority-Based Keyword Search in Databases. A Balmin, V Hristidis, Y Papakonstantinou. VLDB 2004.

[6] A feasibility study and performance study of dependency inference. D. Bitton, J. Millman, S. Torgerson, Proc. ICDE 1989 pg. 635-641.

[7] Effective change detection using sampling. J. Cho and A. Ntoulas. VLDB 2002.

[8] A Fast Regular Expression Indexing Engine. J. Cho and S. Rajagopalan. ICDE 2002.

[9] An Improved Data Stream Summary: The Count-Min Sketch and Its Application. G. Cormode and S. Muthukrishnan. LATIN 2004, pg. 29-38.

[10] Mining database structure; or, how to build a data quality browser. T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. SIGMOD 2002, 240-251.

[11] Grouping multivariate time series: A case study. T. Dasu, D. F. Swayne and D. Poole. Intl. Conf. Data Mining, Workshop on Temporal Data Mining 2005.

[12] Reconciling schemas of disparate data sources: A machine learning approach. A. Doan, P. Domingos, and A. Levy. SIGMOD 2001, pg. 509-520.

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

[14] A Framework for Measuring Changes in Data Characteristics. V. Ganti, J. Gehrke, and R. Ramakrishnan. PODS 1999 pg 126-137

[15] Comparing Massive High-Dimensional Data Sets. T. Johnson, T. Dasu. KDD 1998, pg. 229-233.

[16] Approximate Joins: Concepts and Techniques. N. Koudas, D. Srivastava. VLDB 2005.

[17] SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. W.-S. Li, C. Clifton. Data and Knowledge Engineering 33(1):49-84, 2000.

[18] Schema mapping as query discovery. M. Hernandez, R. Miller, and L. Haas. SIGMOD 2001, pg. 607.

[19] Indexing Text Data under Space Constraints. B Hore, H Hacigumus, B Iyer, S Mehrotra CIKM 2004.

[20] Efficient discovery of functional dependencies and approximate dependencies using partitions. Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. Proc. ICDE 1998, pg. 363-372.

[21] Efficient IR-Style Keyword Search over Relational Databases. V Hristidis, L Gravano, Y Papakonstantinou VLDB, 2003.

[22] DISCOVER: Keyword Search in Relational Databases. V Hristidis, Y Papakonstantinou. VLDB, 2002.

[23] Modeling and Managening Content Changes in Text Databases, P. Ipeirotis, A. Ntoulas, J. Cho, L. Gravano. ICDE 2006, pg 606-617.

[24] Detecting Change in Data Streams. D. Kifer, S. Ben-David, and J. Gehrke. VLDB 2004, pg. 180-191.

[25] Checks and Balances: Monitoring Data Quality Problems in Network Traffic Databases. F. Korn, S. Muthukrishnan, Y. Zhu. VLDB 2003 536-547.

[26] Mining Deviants in a Time Series Database. H.V. Jagadish, N. Koudas, S. Muthukrishnan, VLDB 1999 102-112.

[27] Outliers in process modeling and identification. Pearson, R.K. IEEE Transactions on Control Systems Technology 10(1):66-63 , Jan 2002.

[28] Practical query analyzer. http://pgfoundry.org/projects/pqa/.

[29] Semantic and schematic similarities between database objects: A context-based approach. V. Kashyap and A. Seth, VLDB Journal 5(4):276-304, 1996.

[30] Issues and approaches of database integration. C. Parent and S. Spaccepietra. CACM 41(5):166-178, 1998.

[31] Efficient search of string partial determinations. S. Kramer and B. Pfahringer. Proc. KDD 1996 pg. 371-378.

[32] Potters Wheel: An interactive data cleaning system. VLDB 2001.

[33] Bottom-up induction of functional dependencies from relations. I. Savnik and P. Flach, Proc. AAAI Knowledge Discovery in Databases, 1993 pg. 174-185.

[34] Similarity Systems. www.similaritysystems.com

[35] Efficiently inducing determinations: A complete and systematic search algorithm that uses pruning. J. Schlimmer. Proc. AAAI Knowledge Discovery in Databases 1993. pg. 284-290.