Bellman : A Data Quality Browser

Theodore Johnson and Tamraparni Dasu
AT&T Labs – Research
johnsont@research.att.com
tamr@research.att.com

Abstract: When a data analyst starts a new project, she is often presented with one or more very large databases (containing hundreds or thousands of tables). Extracting useful information from the databases can be a difficult problem: documentation is usually minimal, the data is poorly structured and difficult to join, and the quality of the data is often poor. As an aid in exploratory analysis, we are developing a data quality browser that allows the analyst to quickly gain an understanding of the contents of the tables and their relationships. In addition, the browser serves as a platform for issuing data mining queries targeted towards a further understanding of data quality problems. We illustrate the utility of the data quality browser with several examples.

Introduction

Starting a new and large data analysis project can be a disheartening prospect, because the first task is to understand a new data set. Often, the analyst is given access to a large production database (or a snapshot of it) and asked to produce results. The database can be very large, containing hundreds or thousands of tables. Documentation is usually minimal. The data is often of poor quality – missing data, mistyped data entries, square pegs in round holes, and so on. The task of extracting information from the database is frustrated by both the complexity of the data and data quality problems in the data.

There are many reasons why production databases become complex and disordered. New applications are fit into old data models, new functions require new tables for support, applications are merged, etc. (see [1] for more discussion). While it would be desirable to prevent the disorder from occurring in the first place, the analyst is (usually) not in a position to dictate how the production database evolves. Instead the analyst must work with the data as it is provided (and perhaps make recommendations about database restructuring for data quality improvement).

We are developing Bellman, a *data quality browser* to help the analyst explore and understand a large, complex, and dirty data set. The key idea is to issue *data profiling* queries, store the results (inferred metadata), and present them to the user. Data profiling results are stored in the database itself, allowing the user to issue ad-hoc queries on the inferred metadata. In this paper, we show how even the availability of simple data profiling queries, stored in the database and available for ad-hoc analysis, can simplify the task of understanding the structure of a database and the quality of the data.

Several companies offer a data profiling product (Evoke Software [2], Metagenix Inc [3], Knowledge Driver [4]), for use in database migration or re-engineering. Another related tool is the Integrity Analyzer [10]. While these tools can be used to support the data analyst, we are developing the data quality browser because:

- Our thesis is that a database browser which is enhanced with data profiling and special purpose data mining features is an effective tool for the data analyst carrying out exploratory and data quality analyses.
- We need to address special features of AT&T data.
- We plan to use the data quality browser as a test bed for research in data quality mining.

Data Profiling

Data profiling refers to the process of querying a database to extract information about the properties of its data. An Evoke Software white paper [5] suggests three kinds of data profiling:

- *Column profiling* refers to the analysis of a particular field of a table. For example, statistics such as the number of NULL values, the number of unique values, the distribution of values, the length or range of the values in the field.
- *Dependency profiling* refers to the analysis of the correlation of fields in the same row of a table. This type of analysis includes the automatic determination of keys (collections of fields whose values are unique in every tuple) and functional dependences (relationships in which one collection of fields uniquely determine the value of another field e.g., a zip code functionally determines a state). See [6, 7, 8] for a discussion of academic research on this topic.
- *Redundancy profiling* refers to the analysis of the correlation of fields across tables. For example, this type of analysis can be used to suggest which fields can be used to join two tables.

It is clear that the column, dependency, and redundancy profiling discussed above is valuable for understanding the structure of a database, a pre-requisite not only for data migration but also for data cleaning. However, it is also clear that additional types of profiling are needed for a data quality browser. For example, additional types of information include summaries of temporal changes in the database, the structure of the database (i.e., a collection of tables) and the inter-relationships between databases.

In its current state, Bellman collects only a few types of statistics – the number of rows in a table, the number of unique values of a field, the number of times a field is NULL, the most frequent values of a field, dependencies between attributes, and "signatures" of field values. However, we have found that a browser that incorporates even this limited amount of information and presents it in an interactive manner greatly improves the database exploration process.

Data Browser Architecture

The architecture of Bellman is shown in Figure 1. The browser consists of three components: a GUI, an analysis engine, and a profile repository. The analysis engine issues queries to the data tables (and perhaps to previously profiled information) and interprets the results. The browser profile repository stores profile information that the analysis engine has computed, and also retrieves it for display by the GUI or for use by the analysis engine. The GUI allows the user to issue commands to the analysis engine and to view results. The data quality browser is written in Java, and accesses the target database using JDBC. The analysis engine is written in C++ and accesses the target database using ODBC or OCI.

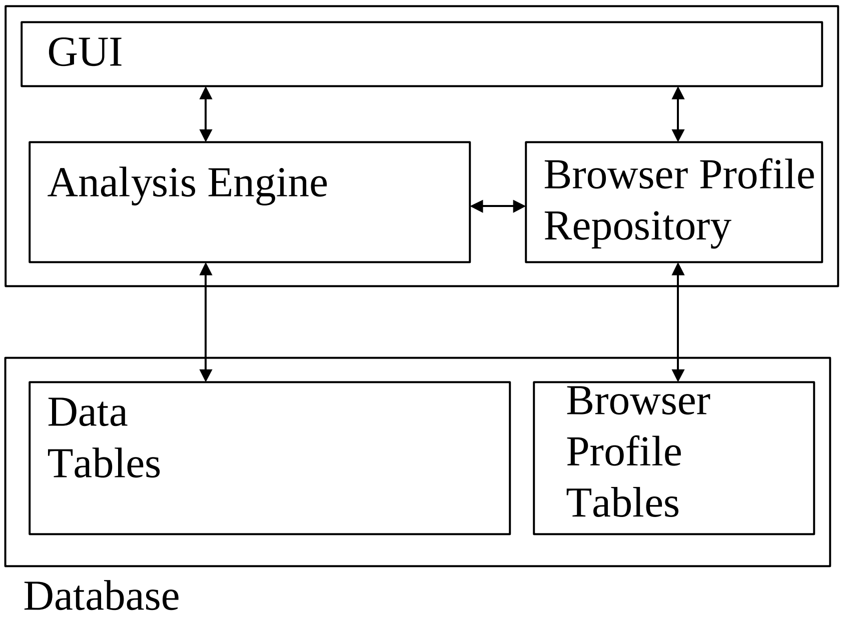
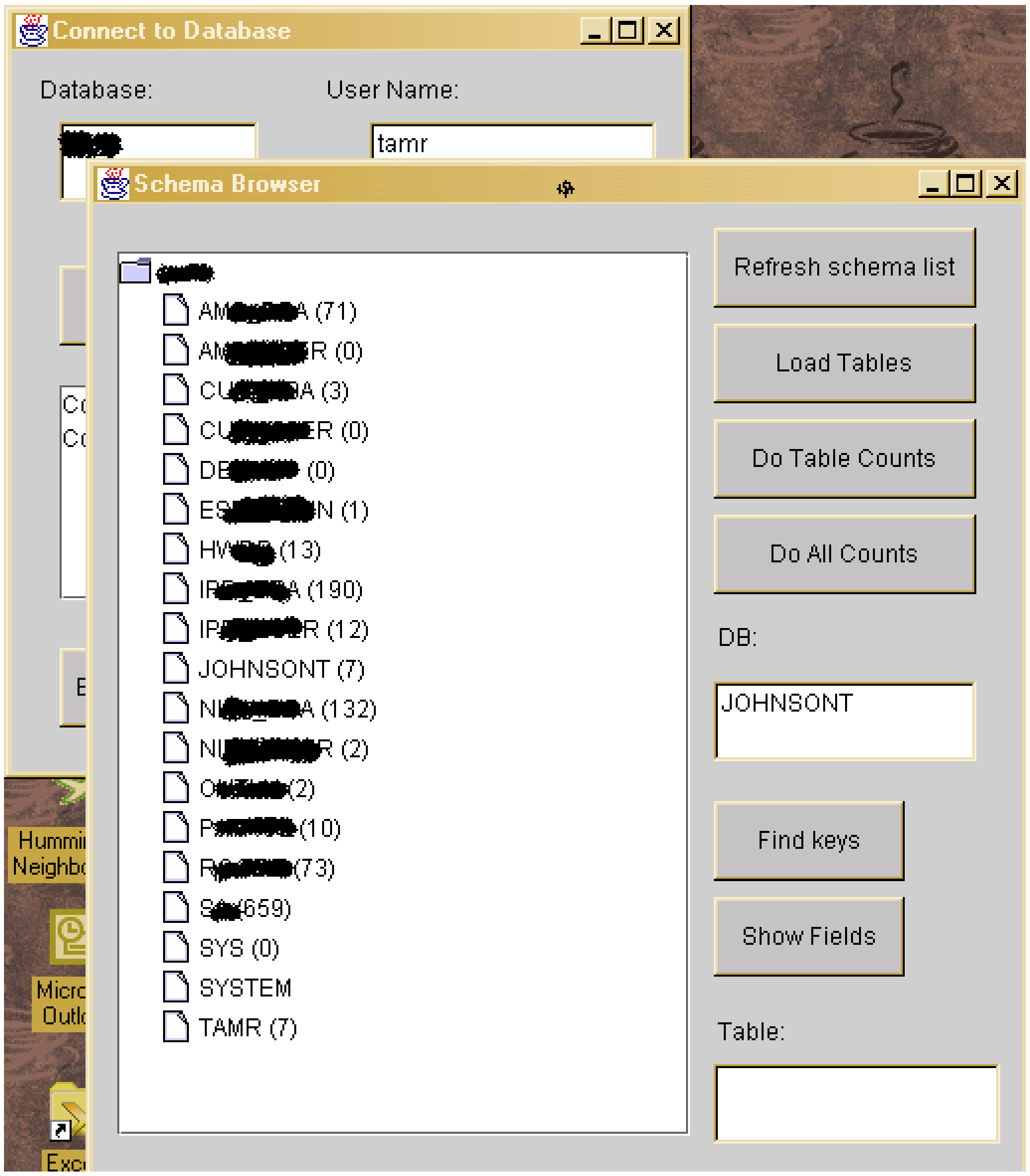


Figure 1 : Architecture of the Data Quality Browser

Example Application

We illustrate the use of Bellman through a series of examples. Figure 2 shows the first screen displayed to the user after connecting to the database (we have censored tablespace names in order to protect AT&T data). The number in parentheses to the right of the tablespace name is the number of tables in the tablespace. Placing this information next to the tablespace name immediately informs the analyst of some tablespace properties. For example, the DB* tablespace has not been loaded and the S* tablespace is very large. From this window, the user can invoke some types of profiling analyses on all tables in a table space (using the “Do Table Counts” and “Do All Counts” buttons).



Database: [redacted]

User Name: tamr

[redacted]

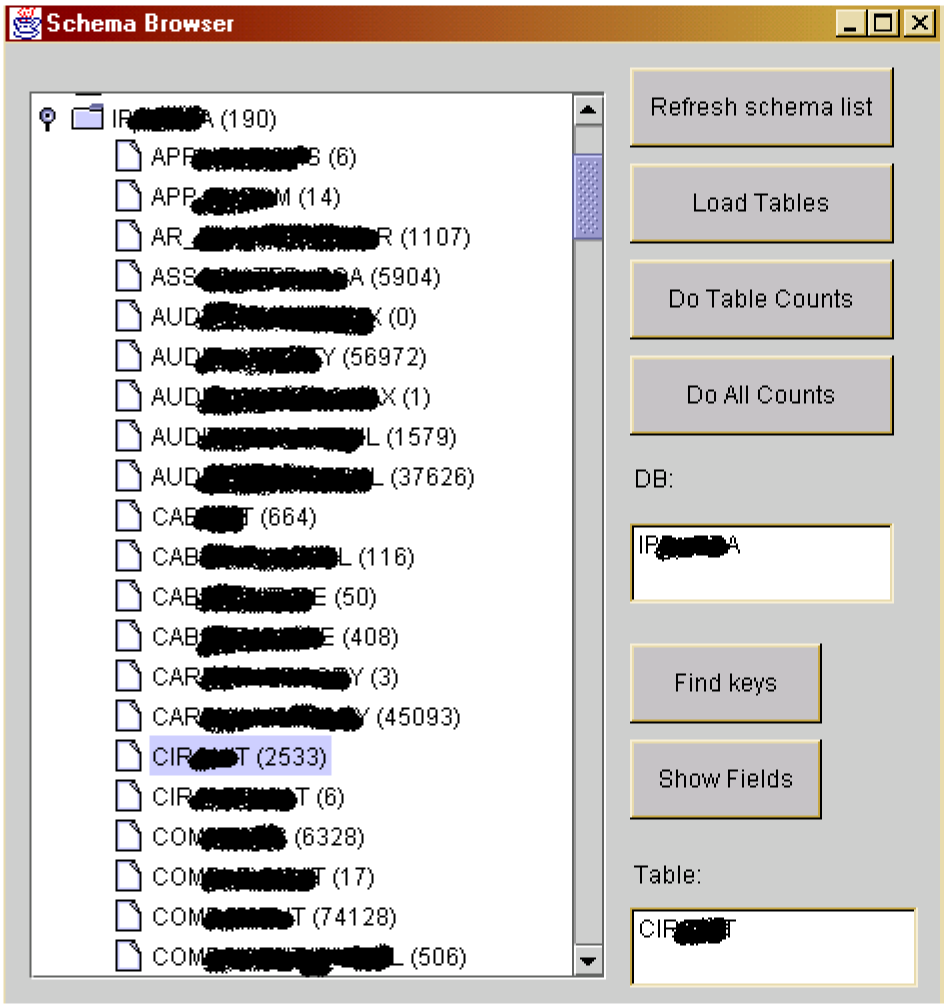
- AM[redacted]A (71)
- AM[redacted]R (0)
- CU[redacted]A (3)
- CU[redacted]ER (0)
- DE[redacted] (0)
- ES[redacted]N (1)
- HW[redacted] (13)
- IR[redacted]A (190)
- IP[redacted]R (12)
- JOHNSONT (7)
- NI[redacted]A (132)
- NI[redacted]R (2)
- O[redacted](2)
- P[redacted](10)
- R[redacted](73)
- S[redacted](659)
- SYS (0)
- SYSTEM
- TAMR (7)

DB: JOHNSONT

Table:

Figure 2 Tablespaces in the databases (annotated with the number of tables in the tablespace).

By clicking on the "Load Tables" button, the user can load all of the tables in a tablespace into the window, as is shown in Figure 3. The number to the right of the table name is the number of rows in the table. This information is useful when browsing a new database as one can see which tables are likely to contain detail data (e.g., very large tables) as opposed to dimensional or special purpose data. We can see that some of the tables are very small or even empty (i.e., the AUD*X tables).



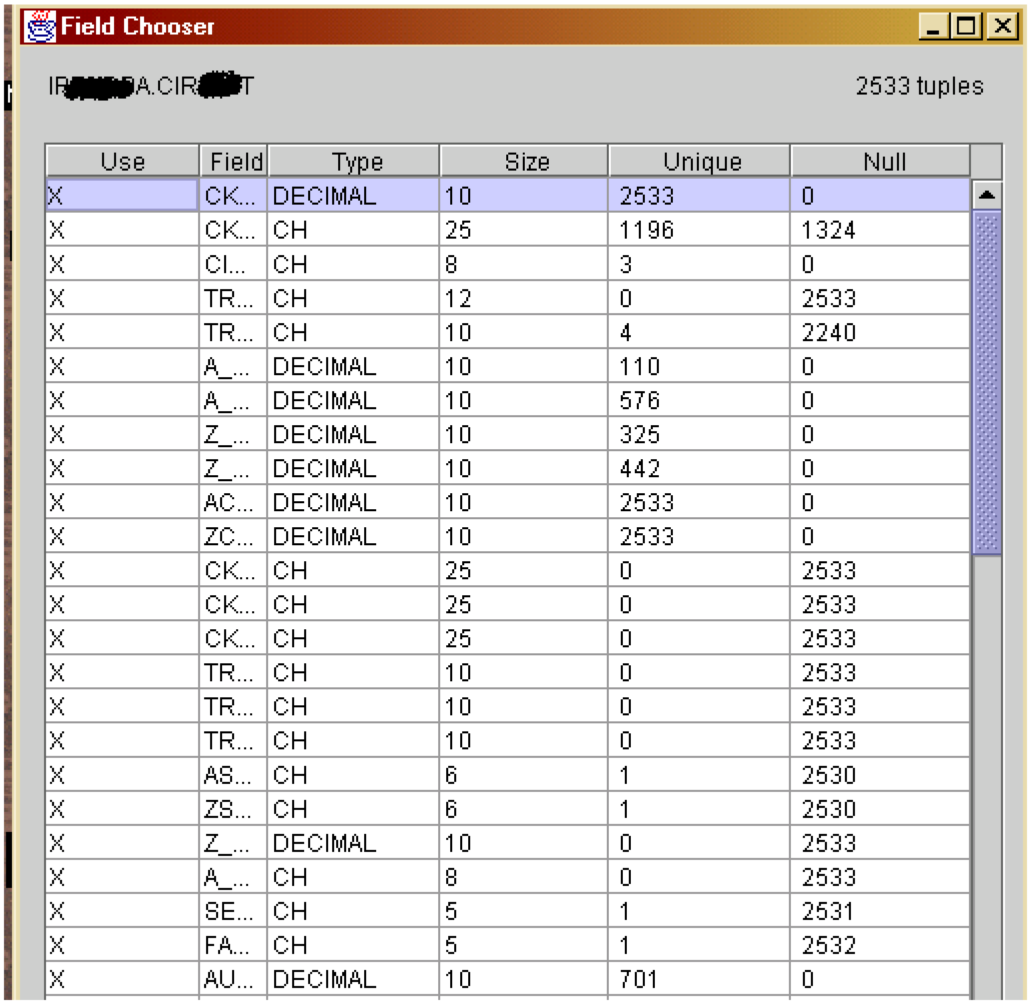
IP[redacted]A (190)
APP[redacted]S (6)
APP[redacted]M (14)
AR_[redacted]R (1107)
ASS[redacted]A (5904)
AUD[redacted]X (0)
AUD[redacted]Y (56972)
AUD[redacted]X (1)
AUD[redacted]L (1579)
AUD[redacted]L (37626)
CAB[redacted]T (664)
CAB[redacted]L (116)
CAB[redacted]E (50)
CAB[redacted]E (408)
CAR[redacted]Y (3)
CAR[redacted]Y (45093)
CIR[redacted]T (2533)
CIR[redacted]T (6)
COM[redacted] (6328)
COM[redacted]T (17)
COM[redacted]T (74128)
COM[redacted]L (506)

DB: IP[redacted]A

Table: CIR[redacted]T

Figure 3: The tables in the IP*A tablespace. The number of rows in the table is to the right of the table name.

We can examine the format of the CIR*T table by clicking on the "Find Keys" button when the CIR*T table is highlighted. A new window, shown in Figure 4, displays the fields of the table along with the field data type, the number of unique values of the field, and the number of null values of the field. The number of rows in the table is displayed in the upper right hand corner of the window. The combination of the number of rows in the table and the number of unique / null values of the field indicates the nature of the information in the field. In this table, it is clear that the 1st, 10th, and 11th fields are all keys. Several of the other fields are likely to contain significant amounts of information, but most of the fields are completely null. Developers often create null fields (and empty tables) as placeholders for the development of new system functions. However these placeholders are confusing to the analyst, hence the value of identifying them.



IR[redacted]A.CIR[redacted]T

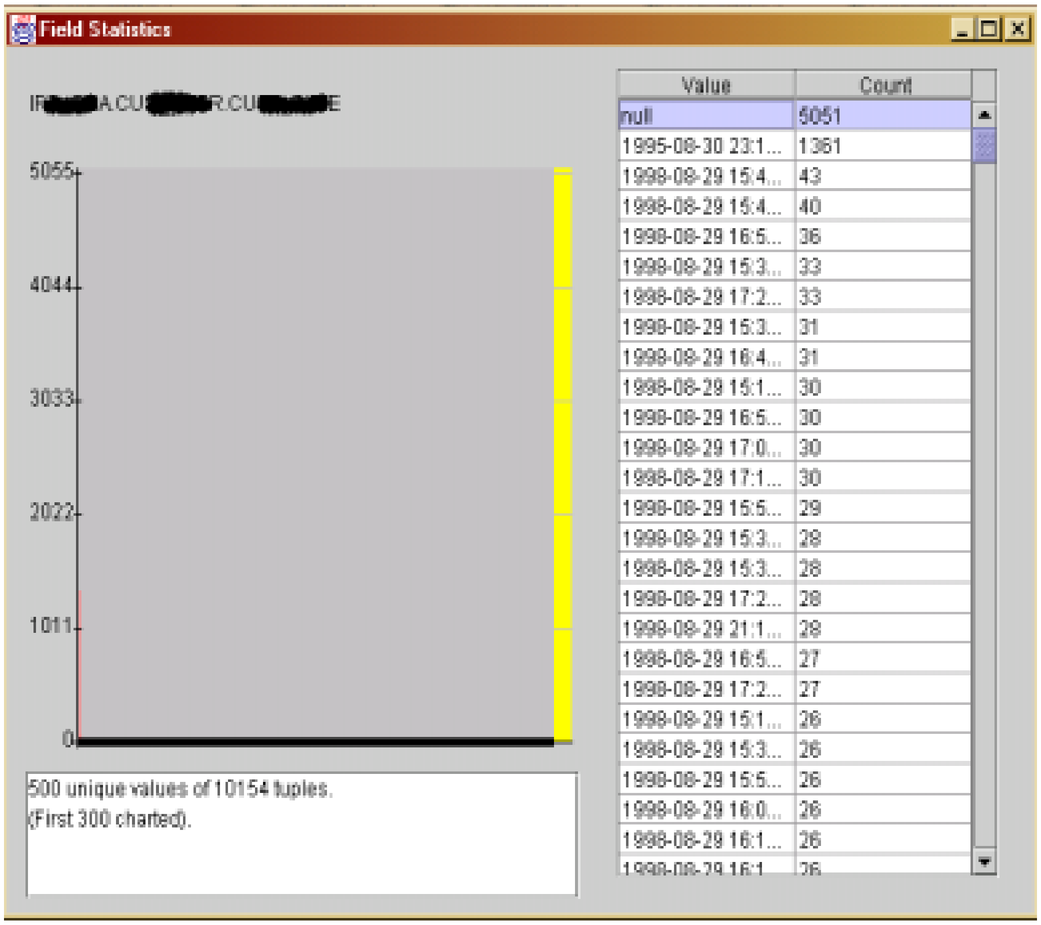
2533 tuples

Use	Field	Type	Size	Unique	Null
X	CK...	DECIMAL	10	2533	0
X	CK...	CH	25	1196	1324
X	CI...	CH	8	3	0
X	TR...	CH	12	0	2533
X	TR...	CH	10	4	2240
X	A_...	DECIMAL	10	110	0
X	A_...	DECIMAL	10	576	0
X	Z_...	DECIMAL	10	325	0
X	Z_...	DECIMAL	10	442	0
X	AC...	DECIMAL	10	2533	0
X	ZC...	DECIMAL	10	2533	0
X	CK...	CH	25	0	2533
X	CK...	CH	25	0	2533
X	CK...	CH	25	0	2533
X	TR...	CH	10	0	2533
X	TR...	CH	10	0	2533
X	TR...	CH	10	0	2533
X	AS...	CH	6	1	2530
X	ZS...	CH	6	1	2530
X	Z_...	DECIMAL	10	0	2533
X	A_...	CH	8	0	2533
X	SE...	CH	5	1	2531
X	FA...	CH	5	1	2532
X	AU...	DECIMAL	10	701	0

Figure 4: Fields in a table.

An option in the "Field Chooser" window will open a new window (shown in Figure 5) that displays the most common values of the highlighted field. Although this is a very simple function, understanding the data in a field is critical for understanding the contents of the database. Making this information available at the click of a button greatly simplifies exploratory tasks.

The chart on the left hand side of the window is a chart of the counts of the most common values. In this case the chart area is almost empty, indicating the extreme skewness of this field. Because default values usually occur frequently, examining the most common values will often expose hidden default values. In this case, the "1995-08-30" date occurs suspiciously often, and is likely to be a default value. The other dates in the window occur suspiciously close together, so they might also be hidden default values.



IR[redacted]A.CU[redacted]R.CU[redacted]E

Value	Count
null	5051
1995-08-30 23:1...	1361
1998-08-29 15:4...	43
1998-08-29 15:4...	40
1998-08-29 16:5...	36
1998-08-29 15:3...	33
1998-08-29 17:2...	33
1998-08-29 15:3...	31
1998-08-29 16:4...	31
1998-08-29 15:1...	30
1998-08-29 16:5...	30
1998-08-29 17:0...	30
1998-08-29 17:1...	30
1998-08-29 15:5...	29
1998-08-29 15:3...	28
1998-08-29 15:3...	28
1998-08-29 17:2...	28
1998-08-29 21:1...	28
1998-08-29 16:5...	27
1998-08-29 17:2...	27
1998-08-29 15:1...	26
1998-08-29 15:3...	26
1998-08-29 15:5...	26
1998-08-29 16:0...	26
1998-08-29 16:1...	26
1998-08-29 16:1	26

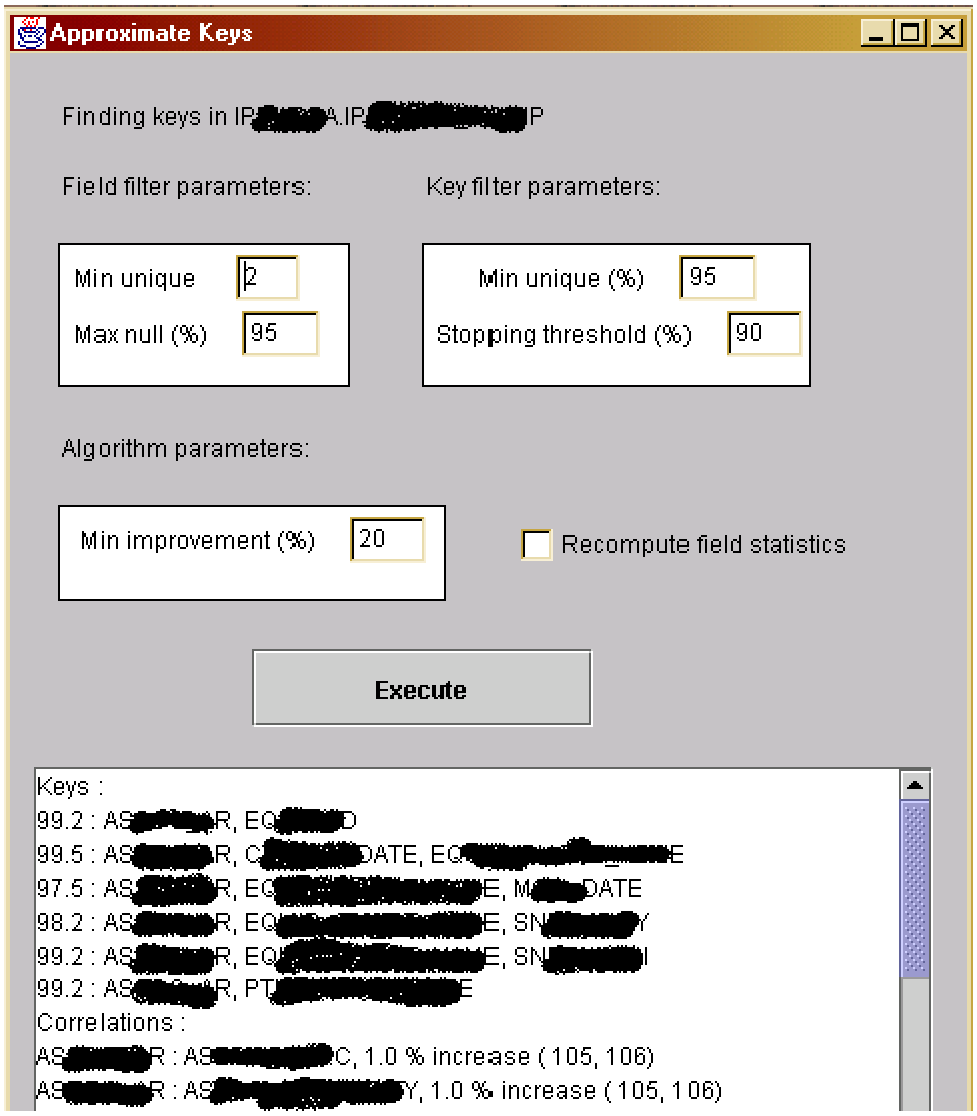
500 unique values of 10154 tuples.
(First 300 charted).

Figure 5: Identifying hidden default values

As Figure 2 shows, the database we set out to analyze is very large and complex. It is also poorly documented. In the process of understanding the structure of the databases, we need to know which fields (or collections of fields) are the keys of the table (that is, they are unique in every row). Because the data is likely to be dirty, we will accept "approximate" keys (i.e., they have few duplicate values).

A window for finding approximate keys is accessible from the "Field Chooser" window shown in Figure 4. Fields marked with an 'X' in the "Use" column are selected as input to the key finding algorithm (we might know in advance that certain fields will not be keys and we can deselect them). Figure 6 shows the key finding widow displaying previously discovered results. At the top of the window is a collection of filtering parameters for the algorithm (to minimize the number of expensive queries necessary to search for a key). The algorithm found two 2-field keys (AS*R and EQ*D, AS*R and PR*E), and four 3-field keys. Note that this table has no single field key.

The key finding algorithm computes the counts of unique values of pairs, triples, etc. of the fields of the table. Therefore as a side effect the key finding algorithm finds *approximate dependencies*, in which one set of fields determines another. The key finding algorithms will not find all such dependencies (because it avoids doing all counts), and does not check the quality of the dependence. These dependencies are marked accordingly when they are entered into the profile repository.



Finding keys in IP[redacted]A.IP[redacted]P

Field filter parameters:

- Min unique: 2
- Max null (%): 95

Key filter parameters:

- Min unique (%): 95
- Stopping threshold (%): 90

Algorithm parameters:

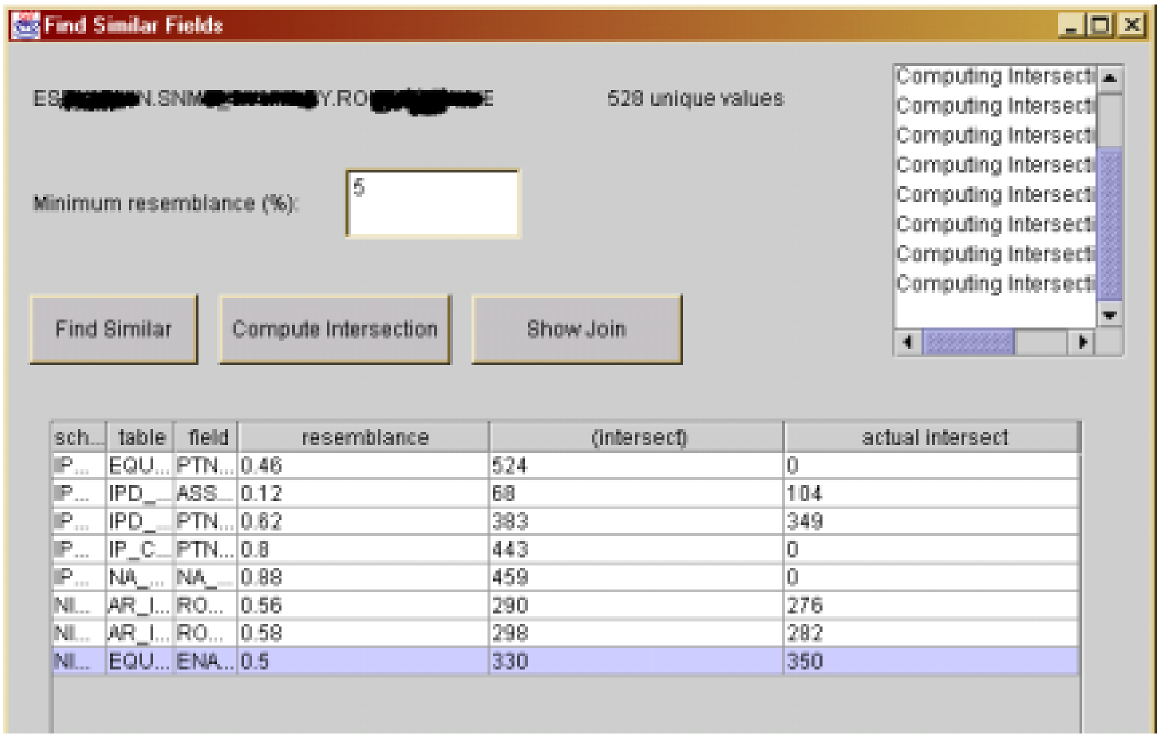
- Min improvement (%): 20
- [ ] Recompute field statistics

```
Keys :
99.2 : AS[redacted]R, EQ[redacted]D
99.5 : AS[redacted]R, C[redacted]DATE, EQ[redacted]E
97.5 : AS[redacted]R, EQ[redacted]E, M[redacted]DATE
98.2 : AS[redacted]R, EQ[redacted]E, SN[redacted]Y
99.2 : AS[redacted]R, EQ[redacted]E, SN[redacted]I
99.2 : AS[redacted]R, PT[redacted]E
Correlations :
AS[redacted]R : AS[redacted]C, 1.0 % increase ( 105, 106)
AS[redacted]R : AS[redacted]Y, 1.0 % increase ( 105, 106)
```

Figure 6: Finding keys and correlations.

Another important element in understanding the structure of the database is the finding of the fields which link tables together (i.e., can be joined). We make use of a *sketch* [9] (a special type of sampling) to represent the set of values in a field using only a few numbers (in the case of Bellman, 50 numbers). By comparing the sketches of two fields, we can approximate the size of their intersection – and therefore whether the tables can be joined using these fields.

In Bellman, we compute a sketch for each field with at least 20 unique values (a tot al of 4700 fields qualified). Figure 7 shows the set of fields found to have a large intersection with the ES*.SNM*.ROU* field. The *resemblance* field contains the value which can be estimated directly from the sketch. Using the resemblance and the number of unique values in the fields, we can estimate the intersection size (in the *(intersection)* column. The "compute intersection" button will compute the actual intersection size, listed in the rightmost column. Some of the actual intersection sizes are computed to be zero, because the fields contain values with the same text but of different data types (in this case, fixed-length character strings versus variable-length character strings). The estimated intersection size is a rough approximation of the actual intersection size, but it is able to distinguish between small, medium, and large intersections. We note that the manes of the similar fields are all significantly different; indicating that the use of field names alone to identify joins paths does not work well.



ES[redacted]N.SNM[redacted]Y.RO[redacted]E 528 unique values

Minimum resemblance (%): 5

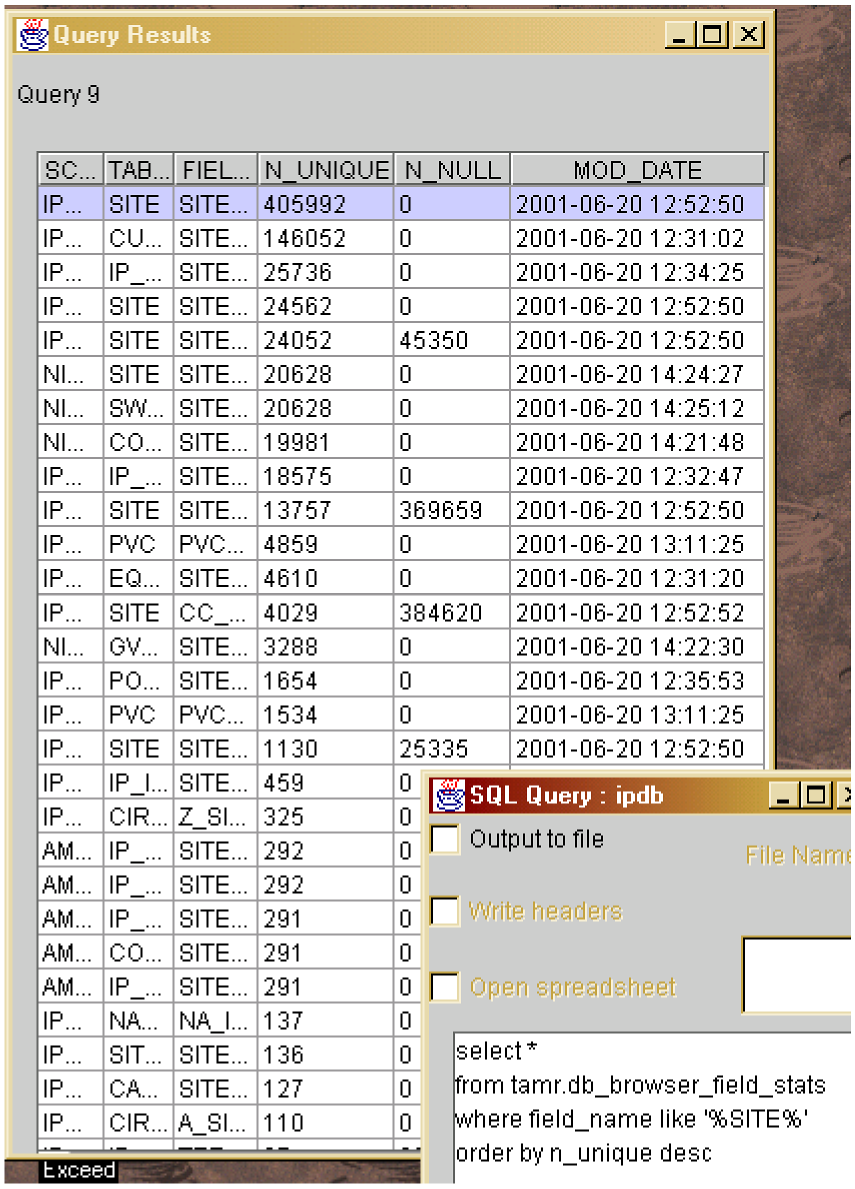
Computing Intersecti
Computing Intersecti
Computing Intersecti
Computing Intersecti
Computing Intersecti
Computing Intersecti
Computing Intersecti
Computing Intersecti

sch...	table	field	resemblance	(intersect)	actual intersect
IP...	EQU...	PTN...	0.46	524	0
IP...	IPD_...	ASS...	0.12	68	104
IP...	IPD_...	PTN...	0.62	383	349
IP...	IP_C...	PTN...	0.8	443	0
IP...	NA_...	NA_...	0.88	459	0
NI...	AR_I...	RO...	0.56	290	276
NI...	AR_I...	RO...	0.58	298	282
NI...	EQU...	ENA...	0.5	330	350

Figure 7. Finding similar fields (an "actual intersection" of zero occurs when the field values contain the same text but are of different data types).

Whenever Bellman performs an analysis task, it stores the profile data in the profile repository. One advantage of storing this data is to make the browser feel interactive – all of the screen shots in this paper appear within a second of the mouse click which requests it. By storing the profile data in the database itself provides another advantage – the profile data is available for ad-hoc queries. In Figure 8, we show a query that asks for fields that might relate to sites, and a portion of the result (the first three fields are the tablespace, the table, and the field). By examining the number of unique values of the fields, we obtain a hint about promising join paths. Other profiled data is also stored in the database and is available for querying – including the approximate keys.

Note that one of the fields in the profile repository table is the date at which the profile data was created. This field allows us to track changes over time.



SC...	TAB...	FIEL...	N_UNIQUE	N_NULL	MOD_DATE
IP...	SITE	SITE...	405992	0	2001-06-20 12:52:50
IP...	CU...	SITE...	146052	0	2001-06-20 12:31:02
IP...	IP_...	SITE...	25736	0	2001-06-20 12:34:25
IP...	SITE	SITE...	24562	0	2001-06-20 12:52:50
IP...	SITE	SITE...	24052	45350	2001-06-20 12:52:50
NI...	SITE	SITE...	20628	0	2001-06-20 14:24:27
NI...	SW...	SITE...	20628	0	2001-06-20 14:25:12
NI...	CO...	SITE...	19981	0	2001-06-20 14:21:48
IP...	IP_...	SITE...	18575	0	2001-06-20 12:32:47
IP...	SITE	SITE...	13757	369659	2001-06-20 12:52:50
IP...	PVC	PVC...	4859	0	2001-06-20 13:11:25
IP...	EQ...	SITE...	4610	0	2001-06-20 12:31:20
IP...	SITE	CC_...	4029	384620	2001-06-20 12:52:52
NI...	GV...	SITE...	3288	0	2001-06-20 14:22:30
IP...	PO...	SITE...	1654	0	2001-06-20 12:35:53
IP...	PVC	PVC...	1534	0	2001-06-20 13:11:25
IP...	SITE	SITE...	1130	25335	2001-06-20 12:52:50
IP...	IP_I...	SITE...	459	0	
IP...	CIR...	Z_SI...	325	0	
AM...	IP_...	SITE...	292	0	
AM...	IP_...	SITE...	292	0	
AM...	IP_...	SITE...	291	0	
AM...	CO...	SITE...	291	0	
AM...	IP_...	SITE...	291	0	
IP...	NA...	NA_I...	137	0	
IP...	SIT...	SITE...	136	0	
IP...	CA...	SITE...	127	0	
IP...	CIR...	A_SI...	110	0	

```
select *
from tamr.db_browser_field_stats
where field_name like '%SITE%'
order by n_unique desc
```

Figure 8: Querying the profile metadata.

Conclusions

In spite of the simple and limited analysis that the current version of Bellman provides, it is already a very useful tool for the exploratory analysis of large, complex, poorly documented, and dirty databases. We have used it to find hidden default values and to explore join paths between different databases (which appear as different tablespaces in our examples). Information about table sizes, and counts of unique and null values allowed us to eliminate many dead ends when searching for join paths. Finding hidden null values is automated by having a pre-formulated query available at the click of a button. More sophisticated analyses will allow us to automate our exploration even further.

We are continuing to Bellman, adding profile data and special purpose analyses to automate the process of exploring new databases and finding data quality problems. Tasks that we plan to support with automation include analyzing textual similarity fields, visualizing the structure of a database, and finding potentially corrupted data.

Bibliography

[1] *Data Quality Issues in Service Provisioning & Billing*, T. Dasu and T. Johnson, submitted to IQ2001.

[2] Evoke Software http://www.evokesoftware.com/

[3] Metagenix Inc. http://www.metagenix.com/home.asp

[4] Knowledge Driver http://www.knowledgedriver.com/

[5] http://www.evokesoftware.com/pdf/wtpprDPM.pdf

[6] *Efficient Discovery of Functional and Approximate Dependencies Using Partitions*, Y. Huhtala, J. Karkkainen, P. Porkka and H. Toivonen, International Conf. On Data Engineering, 1998.

[7] *Dependency Inference*, H. Mannila and K.J. Raiha, Proc. 13th Intl. Conf. On Very Large Data Bases, pg. 155-158, 1987.

[8] *Bottom-up Induction of Functional Dependencies from Relations*, I. Savnik and P. Flach, AAAI Workshop (KDD '93), pg. 174-185, 1993.

[9] *On the resemblance and containment of documents*, A. Z. Broder, Compression and Complexity of Sequences, pg. 21-19, 1997.

[10] *Quality Information and Knowledge*, K.-T. Huang, Y. W. Lee, R. Y. Wang, Prentice-Hall 1999.