OBSERVABILITY OF INFORMATION IN DATABASES
New Spins in Data Warehousing for Credit Risk Management

Stjepan Pavlek and Damir Kalpić
Faculty of computing and electrical engineering, University of Zagreb, Unska 3, Zagreb, Croatia

Keywords: Observability, data warehousing, data mining, free text fields, coding schemes, chart of accounts.

Abstract: In this paper the observability of information in modern databases is investigated. Observable information is the one that is explicitly stored in a database. Unobservable information is information hidden in various coding schemes, transaction streams or free-text-description fields. Nowadays credit risk management tends to employ cutting edge technologies and approaches from the fields of statistics and machine learning for achieving their goals. Often it is forgotten that machine learning schemes can only use completely observable information. The issue is eventually addressed – but instead when using the data it should be addressed during the data warehouse design phase.

1 INTRODUCTION

The data warehousing branch of the computer science had focused on easy information retrieval as one of its top priorities ever since its inception.

In the area of straightforward information retrieval it is possible to identify several layers at which the problem is being addressed. Furthermore, we can notice the focus advancing from first to last of these layers in time.

At first, the focus was on technical issues regarding the organization of the database and the algorithms used to access and manipulate the data in the database. Some of the breakthrough ideas and concepts that would classify in this level are: Professor Kimball's star schema (Kimball, 1996); Hash join algorithm (Bratbergsengen, 1984); Bitmap indices (O'Neil, 1987).

Knowledge involved in making progress at this level is technical, computer science knowledge, and there is none, or minimal, involvement from the side of "business users". We can moreover note the preceding decade (the nineties) and the first part of the current decade being the golden age of this layer.

Second layer can be recognized as focus on the data integration. It has been taking momentum since the beginning of the current decade and is among top buzzwords today. The spiritus movens here is that we can see more about our business; that we can learn more about our customers or that we can better manage our risk if we are able to query data across all of our platforms, across different transaction systems supporting different aspects of our business, etc.

The achievements in this layer can be depicted as advancing through different sublayers within it, as through stages. It is one thing to pull the customer data from different databases into a single database (or to be able to query transparently different databases from a single user interface, with a single user query). Then, it is another thing to have these two "lists" of customers matched so the same customer from the two sources is recognized to be the same. But this is still the beginning: furthermore, we need to have the attributes (which means mainly the domains of the attributes) unified across the sources. Then there are hierarchies and classifications built on top of customers, and so on.

This is the story of the data integration (short version). To operate in this area it is required to comprehend considerable understanding of the data in business terms, either in a person or in a team. We need to understand the attributes describing certain database entity, to know what exactly the meaning of its domain values is, in order to be able to recognize when the same thing has been given two different names, or vice versa, when two different things are being called the same name.

Pavlek S. and Kalpić D. (2008).
OBSERVABILITY OF INFORMATION IN DATABASES - New Spins in Data Warehousing for Credit Risk Management.
In *Proceedings of the Third International Conference on Software and Data Technologies*, pages 361-368
DOI: 10.5220/0001892903610368

2 DEFINITIONS AND PROBLEM SETUP

After the data integration and surrounding concepts (like the data cleansing) the focus shifts to the next layer at which we can consider the ability to retrieve information from a data warehouse. This level doesn't yet wear a well-known brand name, like Star schema, or Data integration, but for this article we've given it the name Observability of information. Making advancements in this level won't require merely to perfectly understand the way the business sees the data and its structure. We will instead see our opportunity in initiating change in the way the business sees the data, in the way the business does business, at least as far as its information organization and information keeping side is concerned. If this sounds somewhat vague or confusing, that's good; it means you need to read the rest of the paper in order to clear thing out.

To wrap up the introduction and the first paragraph of this section, we can say that thinking in this layer of observability of the data is the topic of this paper.

2.1 The Term Observability in other Disciplines

The term observability originates from the control theory where it is a measure for how well internal states of a system can be inferred by knowledge of its external outputs (Wikipedia, 2008)

In software industry the term has been used to describe the lack of ability to assess which code exactly is being executed (what's going on) in complex software environments. In conjunction with software layering (which is a technique that enables us to e.g. program an application without programming as well a database engine, an operating system, etc.) the term observability is used to describe the problem of not being able to observe the unintentional actions that are provoked in lower layers of software stack while programming on a higher abstraction layer.

When mentioned in context of databases, observability again refers to software observability with the meaning of being able to observe what's going on in a system.

The term "data observability", besides being used in control industry, is used in machine learning to describe how much of the data on which the learning is based is observable to the machine learning algorithm.

"Information observability" and "Observability of Information" had been endemically used in economics, sociology, risk management, etc.

2.2 Observability of Information in Databases

Term "Observability of information in databases" (or similar) couldn't be found using internet search engines therefore we assume it hadn't been in use yet.

With this term we want to go along with the known "data" versus "information" versus "knowledge", as established in the data mining terminology, where each of succeeding terms implies more context, more meaning and structure.

For instance, we could have stored data in a database saying that customer A has value for "customer code" equal to 3. To a business user who understands the data, this data, 3, can for instance give information that customer A resides in a foreign country. Because all domestic customers have codes 1 and 2, and all foreign customers have codes 3 and 4.

Let us define that in cases like this the information (regarding residence) is stored in a database – because it is possible to determine for each customer in the customers table whether it is domestic or foreign customer. Also, let's define that the information is not explicitly written because in order to determine the residence for each customer (or any customer in this case), one needs to know "something more" – the meaning of codes 1, 2, 3 and 4.

Definition of observability can now be given as: Information that is not explicitly stored in a database is less observable than the one that is explicitly stored.

3 EXAMPLES OF UNOBSERVABLE INFORMATION

The definition of observability in previous section has been given through the obviously scholar example. Our goal in this section is to enumerate typical situations from the real world in which the information in databases is stored in unobservable manner.

We give three such classes here, but without an aspiration for the list to be exhaustive. Simply, the aim of our research is not to thoroughly cover all flavours in which unobservable information could

appear, but rather to raise awareness of the issue and to offer the idea of how to handle it.

3.1 Free Text Fields

We start with the easiest one and the most obvious example, but there is a special characteristic we want to point out here.

Free text fields are, of course, necessary in every database and in every data warehouse as well. But its usage should be strictly limited to data which by its nature is "free text" and doesn't have structure, like names.

3.1.1 When it Occurs

Free text for storing structured information in an unstructured manner usually occurs when developers of source, transaction, systems judge that some information is applicable only for small percentage of entities in a class (rows in a table), that it is too complex to model it properly, and – this one is decisive – it won't be used programmatically (in code) so there is no point in bothering to develop support for storing it in a structured way (which could include adding fields to existing tables, creating new tables in a model, etc.).

When such decision is made it is usually wrong because of at least two reasons. The first one is that it will be used programmatically in data warehouse, if it is not in transactional system. Someone will want to group by this data point, filter by it, etc. otherwise they wouldn't ask it in the information system at all in first place. Second reason is: it is not being used programmatically today, but there is a new feature request coming tomorrow.

3.1.2 Amending the Definition of Observability of Data

And the characteristic that was mentioned in the beginning: in the definition of observability it was said that the information is unobservable if there is "something else" one needs to know in order to understand the information. With free text fields this often isn't the case.

For instance, if the information about customer's incomes isn't stored in the designated field titled "income", but instead the value "Income = 5.000" is stored in the field "comment", there is now extra knowledge one needs to retrieve the information from the data written in a database. All needed is already written in the comment field. The hatch is that the "instruction" on what actually is contained in the text field is not readable by a computer; it is readable only by a human.

If we would want to be very strict, we would need to amend our definition of observability in order to cover this case.

3.2 Transaction Streams and Similar

Second example we'll look at are transaction streams. Suppose there is a table with all transactions that were done against one loan account in a bank. In this data it is also written all behavioural information one could ask about this loan: which amount is due but not paid, what is the greatest number of days in history of this loan that the due amount was outstanding, what is the average time-weighted past due amount, and so forth.

3.2.1 Comparing with the Free Text Fields

This example is somehow positioned on the opposite pole then the previous case of free text fields. While in the case of free text fields the information hidden in data was obvious to human consumer, but unintelligible to the computer, in this case the information is impossible for a human to read out, and computer-crunching is the only way to get to it. Still, there is a far way to go from computers "needing" to extract this information to them actually doing so.

This situation can appear in two ways. One is when it takes extraordinary programmers' effort in order to design algorithm which gives accurate calculation of what was asked. Example of this would be to calculate e.g. how many days in total the loan had outstanding due amount. (Perhaps this doesn't sound that complicated, but in real world situation it would take months to crack this one.)

Second way is when the algorithm is trivial, but is computationally highly demanding, thus still representing significant implementation problem. Example of such problem would be to calculate the average balance in previous year for each account in a bank, from the table of all bank's last year transactions.

3.2.2 Comparing with the Data Mining

The approach of extracting information through waste number of calculations against raw data is exactly what data mining does. There is a difference though between a problem class we talk about here and the class of problems that are usually addressed through data mining (or classic statistics) techniques.

In theory, appropriate data mining scheme would find a way to calculate some complex measure,

given this measure is relevant for the target attribute (e.g. the probability of default for a customer). In practice it is unlikely that any data mining algorithm would be successful in case of very complex calculations. It would be unacceptably suboptimal to leave such calculations to "good fortune", instead of doing them explicitly.

Therefore, there is no case for skipping to generate complex attributes (calculated measures) for which is known to be relevant.

3.3 Coding Schemes

The third case mentions the coding schemes in context of observability versus unobservability of information in a database.

The example is imperative of this paper because, in contrast to first two examples, coding schemes are in general rarely recognized as structure that is aggravating the information retrieval rather than rendering it easier.

3.3.1 What are Coding Schemes

Vocabulary definition of the term "coding scheme" says: Coding scheme is a set of rules that maps the elements of one set, the coded set, onto the elements of another set, the code element set (Institute for Telecommunication Sciences, 2000). The term is mostly used in the telecommunications science.

In databases, coding scheme would be an entity which is used to group, or hierarchise, instances of some other entity. The elements (table rows) of coding scheme typically consist of fields "code" and "name". Code is usually a set of numerals and name is the text field describing the meaning of the code.

For instance, let's take our previous example of a customer attribute "customer code", with values 1, 2, 3 and 4; where 1 and 2 were resident customers and 3 and 4 are foreign. If we would define a table which would have four rows, numbers one to four in one column and descriptions, e.g. "Resident natural persons", "Resident legal persons", "Foreign natural persons" and "Foreign legal persons" in second column, we could say we've defined a coding scheme.

Often in coding schemes the hierarchy is reflected already in the way codes are composed, defined. So, in our example it would be typical to have codes 00, 01, 10, 11 – instead of 1, 2, 3 and 4. Then it could be defined that position one defines residence (domestic – foreign) and position two defines whether customer is legal or natural person.

3.3.2 Why Coding Scheme is Bad Modelling

In the definition of unobservability it was said it is unobservable that the value 1 carries the information about customer being resident natural person, because the one querying the database needs to know "something else" – the meaning of codes.

Seams that if we create the coding scheme table we've solved the problem – the information is stored in a database. Here is how it is: if we define two attributes, one named "residence", with values "Foreign" and "Domestic" and another with values "Legal person" and "Natural person", we've done a good job and we hadn't introduced the coding schemes problem. This is not the kind of a coding scheme we are trying to depict here as being problematic.

The example of four values is minimalistic example on which the idea was constructed. In reality coding schemes grow to hundreds or thousands of rows. Then the name of a category doesn't say "domestic" or "foreign". It can say something like: "Foreign natural person customer that is under supervision of KS department and also has relation with ABC bank in France." So, the problem with coding schemes is that the problem of no information in database got "solved" in the way that the information is modelled in a free text field. That is slightly better than no information – there is for instance smaller probability for the needed piece of information to get lost – but is still quite inadequate, for the reasons discussed in subsection about free text fields.

3.3.3 The Grand Coding Scheme

And now we are arriving to the greatest coding scheme that one can encounter in a bank. It is the chart of accounts. Chart of accounts is a list of all accounts tracked by an accounting system. Wikipedia further explains chart of accounts as: "should be designed to capture financial information to make good financial decisions". This definition gives a good idea about what chart of accounts *used* to be.

Chart of accounts is a hierarchical structure of codes that group material value (either balances or transactions). At top level the grouping is usually done into: assets, liabilities, equity, income and expenses. Each of these categories further breaks down into company's products or services, types of counterparties to which the product or service is related, and numerous other attributes, like: information about terms, currencies, adjustments, risk provisions, etc. The list is virtually indefinite. For any classical accountant this is a must, a cornerstone.

In order to understand their perspective we need to do some time travelling. Let's go for example back to nineteen sixties and imagine how business reporting might function at that time. Today, we crunch millions of rows of analytical data about customers, or transactions, using large number of attributes for filtering and grouping the data. But what could we do without computers? The answer is simple: there is only a finite, rigorously limited, and relatively small number of groupings we can do: we can group only on attributes that are included in the chart of accounts coding scheme.

We can easily show the total amount of assets or liabilities the company holds. We can further see how much is in short term instruments or deals and how much in long term, given short-long term is the next category that divides the accounts. But, for instance, if there is no special account for short term loans given to banks and a special account for short term loans given to companies, but only one account for all short term loans, there is no way to give out those two figures in a report. It could be done only with going back to analytics and then getting to those numbers summarizing transaction by transaction. At that is unrealistic to do without the computers.

Of course from nowadays perspective, when we can always go back to analytics and issue a GROUP BY statement on any attribute and (relatively) quickly get a result, all this is nonsense. We could just ignore the chart of accounts and everything about it. But it is not simple to do that at all. We should bear in mind that this is one of the oldest surviving management tools. The way the accounting is done hasn't much changed since the double-entry bookkeeping was first introduced by Benedikt Kotruljević of Dubrovnik Republic back in 1458 A.D. (Phillips, Brook, 2003). It is understandable that something surviving for so long is not so easy to kill.

The problem is that entire system is built around these accounts. The way businesses, banks function are built on top of chart of accounts. Neural networks in human brains are adapted to this perspective of the world. Practically, these are the issues:

1. The data included in the chart of accounts doesn't exist outside of it – there was no need to develop model in database in order to support handling the data that is "already included" in chart of accounts;
2. If there is data outside of account codes, it is wrong or unreliable – since all the reports are based on the accounts it was hard to keep discipline and maintain the accuracy of the data which is anyway not being used;
3. It is not actually known what the business demands are outside of accounts, business people "speak in accounts". The report specification doesn't say: "summarize the balance of due interest amount of all loans to B type customers". Instead, it says: "summarize the balances of all accounts that start with numbers 3, 4 or 5, that have numbers 1 and 0 on fourth and fifth place, but don't end with 7, or have number 2 in sixth place …"
4. The entire system is built on accounts. It works (yes, but how?). It would be too big change, it's impossible.

These are the main complications that keep accounts alive. Not just alive, a central reporting tool.

4 SHORTCOMINGS OF DATABASE WITH UNOBSERVABLE INFORMATION

We've used the central part of the paper for explaining which cases we call "cases of unobservable information", where these cases appear, why they appear, and why it is hard to eliminate them. Now let's spend some paragraphs on describing what exactly the impact of unobservable data on our system is. Again, as when listing the different cases of the unobservable data, we want to depict a couple (three) of such cases, but without an attempt to be exhaustive.

4.1 Relying on Accounts in Reporting

That's why they, the accounts, were created in a first place – to support reporting. So why should this be a problem?

If we remember the point three in previous section, where the report specification was given through the "language of accounts", we can imagine there will be countless such specifications in modern organizations. And it is becoming huge burden to maintain them.

Some decades ago reporting was not as dynamical as today. There was a set of standard reports that needed to be done every so, period. Today, there is an explosion of demands for ever new and ever more complex reports, both from regulatory authorities and business users. Continuously, the scope of what

is being reported on expands. How does that change the role of chart of accounts in reporting from being a great tool to being a nuisance? Let's construct a small case study.

Suppose there is a report in a bank that groups balances according to the classification of customers that hold those balances. Since chart of accounts the bank uses first groups by products, and then by customer types, each customer type will appear repeatedly in different branches of hierarchical tree of accounts, once under each product. So the list of accounts that should be taken into consideration for the report is "number of products" times "number of customer types".

Now let's suppose a new product is added to bank's portfolio. In chart of accounts, a new branch will be created for that product. In it, accounts will be created for each customer type. The list of the accounts in specification of the mentioned report needs to be amended. This is the case.

Projected to real world: because of new reporting demands, business growing, mergers, etc, chart of accounts – which perhaps once was a stable structure – became *live* and is changing daily. There are thousands of lists of accounts that need to be amended (in worst case) every time the chart of accounts is changed.

Now let's take a look at alternative design. The report hadn't been implemented through a list of accounts, but instead it relies on CUSTOMER table's "customer type" attribute. In order to prepare the report the SQL join needs to be performed between table ANALYTIC_BALANCES and table with customers. The result set needs to be grouped by the customer type attribute, while the AMOUNT column needs to be treated with SUM operator.

Bank starts a new product. New row needs to be added in PRODUCTS table. There is no impact on aforementioned report whatsoever.

The practical consequences of difference between these two approaches are immeasurable.

4.2 Code Stability

Often, there is code which executes conditionally, depending of the value of one or more attributes of the data. For instance, interest rates need to be calculated and then debited or credited to correct accounts; warnings need to be sent to owners of outstanding past due debts, etc.

Again, as with the reporting, exact specification on how this should be done rely either on account codes, or involved-entity attributes. All considerations mentioned while discussing reporting hold.

In both cases there is inclination towards making errors, if the "relying on accounts" principle is being practiced. Not every time every list of accounts that is impacted will be correctly amended. With time the situation deteriorates. The lists that were supposed to specify the same groups of accounts (the same amount of money in the end) won't do so, because in time there were different mistakes done in maintaining those lists.

Regarding the inclination towards errors, it is notable that mistakes done by the code that manipulates data (like automatic booking of interest) are more grave then the ones that *only* produce errors in reports.

4.3 Data Mining Algorithms

It is impossible to use any information that is hidden in text fields or in coding schemes for data mining. Because of the nature of data mining algorithms it is preferable to keep every independent piece of information in a separate, well defined, attribute – table column. (Witten, Frank, 2000)

What has just been said is notorious to anyone who ever dealt with data mining in practice.

However, we must understand the business users could see things somewhat differently. For instance, they could infer: through chart of accounts the information A is clearly specified. The consultants, experts on data mining, told me that if this information is available, we can reach the demanded goal.

"Data mining algorithms" might be a too narrow title for this section. Besides data mining algorithms there is a wider field of modern trends of automating decision making or support for decision making. Balanced scorecards that are being developed for monitoring the performance of the business, credit scoring scorecards for automating processes of loan approval and other similar techniques don't necessarily rely on some machine learning scheme. But all those modern techniques have in common that they cannot make use out of unobservable data in a data warehouse.

5 PATHS TO SOLUTIONS

Solving the problems described in this paper will surely take much more work from the wider pond of researchers and people that encounter these problems in the industry.

However, as we were encountering them in our work, we began to treat them and we've tried out a couple of strategies and approaches to combat these

problems and annul their consequences. Therefore, it is probably worth sharing some of these approaches that might have potential.

5.1 "Atributising" Coding Schemes

It has been said before that it is unpractical to simply abandon coding schemas, and the strongest reason is because too much has been built on them.

Huge practical problem is incompatibility of two ways of constructing logics. If we would simply replace logic built on chart of accounts with logic built on attributes of customers, contracts, balances and other entities it would be extremely hard to compare the old and the new ways in order to check for errors in new methodology and clear out those errors.

We've come up with a two stage approach that chains two logics and builds attributes based logic on top of chart of accounts logic! This might sound awkward but it actually works great and already improves the model significantly.

So, what's been done: We identify a piece of logic, let's take example from earlier: a report that groups balances according to the classification of customers that hold those balances. There is an old logic in place: a list of accounts connected to each category in customer classification is maintained.

The next thing we do: we implement a new attribute in the chart of accounts table, being the customer category for which the account is meant. We (the accounting department actually) fill in the values for the new attribute for each account in chart of accounts table.

Now we replace the old logic of list of accounts with the new logic built on new attribute in the accounts table. We can directly compare new logic with the old one – because the new logic doesn't select only the list of customers, it selects the list of accounts as well, and two lists of accounts can be directly compared.

And the major benefits of switching logic have already been achieved: if new accounts are opened for new products, in the process of opening those correct values for the new attribute "customer type" will be chosen and there will be no need to amend thousands of lists of accounts. Secondly, as the logic has been switched to an attribute that specifically says the account belongs to "group A" there is no more danger that new account will be added to group A accounts in one list and not added in another – the list is defined as "list of accounts that are group A" and the new account will, depending whether the attribute is set or not, be either added to group A in all lists, or none. This is much better than the case when error can appear in only some lists, because that way it's much more probable that it will stay unnoticed.

The process ends when for each piece of information hidden in chart of account codes a separate attribute in chart of accounts table has been created. This at the same time means that all unobservable information from chart of account codes has been exposed in explicit attributes.

We are not far now from switching from these newly created attributes to *natural* attributes (natural place for the customer type attribute is still customers table, not chart of accounts table). But this still might not happen because of, for instance, legal reasons: still, in many of the countries, if not all, the regulations demand the booking and reporting to be done based on the chart of accounts, therefore we cannot throw it away.

5.2 Decomposing Coding Schemes

Second technique we might want to employ is splitting one coding scheme into more. Coding scheme that groups e.g. customers both into geographical hierarchy and industry they are operating in could be split into two hierarchies: one for geography and another for industries.

In this way the step from having attribute with unobservable information we can move towards having attributes with observable information.

5.3 Pruning Coding Schemes

If an entity, e.g. "contracts" has several attributes, one of which is coding scheme and other are nice, observable, nominal attributes; further, if some information is contained both in the codes of the schema and again in some of the independent attributes; then coding scheme should be pruned in order to exclude repeated information.

Suppose we have coding scheme "products" attached to entity contracts. It divides the contracts into loans, current accounts, guaranties, etc, but it also divides them into short and long term, domestic or foreign account or similar. So a single code from the coding scheme can have the meaning like: "short term loan in local currency".

The information that the loan is in local currency exists in two places: in the code and in separate attribute "currency". What we can do in this case is restructure the coding scheme in such way that it is impossible to read out from it any information which is stored in other attributes.

Or, for practical reasons, we might want to create new coding scheme which includes only information not contained in other attributes.

6 CONCLUSIONS

We've started with very wide considerations on issues of easy information retrieval from data warehouses in introduction section. After naming technical and data integration aspects we've focused on new topic of consideration and called it "observability of information in databases".

After a discussion on this new topic, describing, defining and viewing it as a whole in section two and first two parts of section three, we've moved in last part of section three to a particular niche within observability topic where we see the biggest benefits could be achieved. It is the area of coding schemes and mostly concentrating on one specific: chart of accounts.

The reason for this is that it is still the central point for reporting and analyses in today's banks, although it is huge break in processes of moving to analytics, as presented in section four.

In section five we've presented a viable approach that gives possibility to effectively cancel out negative aspects of unobservability of information in the chart of accounts. At the same time, this approach leaves the possibility to keep chart of accounts in place in which it needs to be due to current regulations.

REFERENCES

Bratbergsengen, K., 1984. *Hashing Methods and Relational Algebra Operations.* Proceedings of the 10th International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc.

Institute for Telecommunication Sciences, 2000. *Telecommunication Standard Terms Glossary.* 2006 edition.

Kimball, R., 1996. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses,* John Wiley & Sons.

O'Neil, P., 1987. *Model 204 Architecture and Performance.* Proceedings of the 2nd International Workshop on High Performance Transaction Systems. Springer-Verlag, London.

Phillips, T., Brook, S., 2003. *The Romance of Double-Entry Bookkeeping,* American Mathematical Society, https://ams.org/featurecolumn/archive/book1.html

Witten, I., Frank, E., 2000. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations,* University of Waikato, Morgan Kaufmann Publishers.

Wikipedia, 2008. http://en.wikipedia.org/wiki/Observability