Deductive Data Warehouses and Aggregate (Derived) Tables

Kornelije Rabuzin, Mirko Malekovic, Mirko Cubrilo
Faculty of Organization and Informatics
University of Zagreb
Varazdin, Croatia
{kornelije.rabuzin, mirko.malekovic, mirko.cubrilo}@foi.hr

***Abstract* - In one of our previous papers, the idea of deductive data warehouses has been introduced. It was shown how to use Datalog rules to perform Online Analytical Processing (OLAP) analysis on data. In this paper, we show how to use Datalog rules to specify the data warehouse model (data mart) as well as how to add rules that produce aggregate and derived tables that are normally used to speed up the process of retrieving data. Since it is good to have aggregate and derived tables (to speed up queries), the main drawback is that they require extra storage. Consequently implicit definition of such tables may seem interesting.**

Keywords: data warehouse; deductive data warehouse; Datalog; aggregate tables; derived tables; data mart.

I. INTRODUCTION

Data warehouses are popular due to the fact they can efficiently store large amounts of data and data can be analyzed by means of front-end business intelligence tools (Business Objects, QlikView, etc.) that support different ways of analysis including drill down, roll up, slice, dice, etc. One can find many different definitions of what a data warehouse is. According to Kimball et al. [2] it is "a system that extracts, cleans, conforms, and delivers source data into a dimensional data store and then supports and implements querying and analysis for the purpose of decision making."

Over the years many companies implemented many (partial) applications and/or information systems that covered only one aspect of business. This was a common scenario in the past any many companies have similar problems today because of such an approach. The main problem is that these companies possess many heterogeneous applications and information systems that were built by means of incompatible technologies (different programming languages were used as well as different ways of storing data) and it becomes hard (expensive) and sometime almost impossible to integrate data from all sources. Now, one may ask why data integration is important? The answer is obvious, especially today, when we know that data must be integrated and compared in order to make good decisions and in order to understand what is really going on.

When building a data warehouse many different steps have to be carried out, but one process that is quite crucial is the so-called Extract Transform Load (ETL) process. The goal of the ETL process is to extract data (from different (non)relational sources), to transform data (data has to be cleaned, business rules have to be obeyed, missing values have to be found, some attributes have to be merged, some values have to be split, different formats have to be unified, etc.) and to load data into the data warehouse. A few good books [3][4][5][7] have been written on data warehouses and ETL; here we just want to emphasize the importance of the ETL process but we will skip the details. All information technology (IT) people, especially those who were working on data warehousing projects, know what end users are capable of doing and what mistakes and bad data transactional sources do often contain.

Since the ETL process is very important, we tried to develop a web ETL tool that had an educational component as well. The main idea was to build a tool that could help users to build a data warehouse and to guide them during the ETL process. More details about the tool can be found in [6]; this is a paper that is under a review at this point in time.

When we talk about the data warehouse data model, we distinguish two types of tables (Fig. 1):

- Fact tables contain facts, i.e., numbers that are used to quantify business processes (number of sold items, number of items in stock, etc.). They contain much larger number of records and small number of attributes.
- Dimension tables (dimensions) contain much larger number of attributes that are used to analyze data in different ways and much smaller number of records (when compared to fact tables).

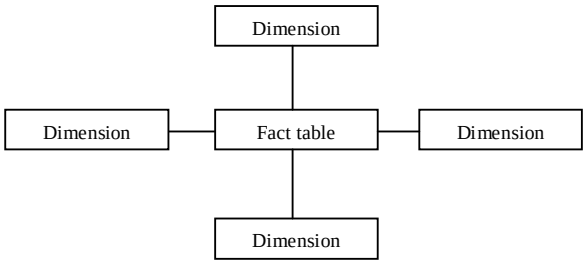

Figure 1. Star schema

When we discuss different data loading techniques, one should keep in mind that data loading is different and it depends on whether a data warehouse already contains some data or not. We have to take care about dimension and fact tables and we have to achieve some level of parallelism, if possible. Because of that we distinguish:

- initial load (a data warehouse is empty and all data have to be inserted),
- incremental load (we change existing records and add new ones) and
- complete reload, i.e., refresh (some or all data are deleted and re-loaded again).

 ISBN: 978-1-61208-346-9

Deductive databases are databases that use deductive rules to produce new information. They usually contain a set of facts (they would correspond to rows in a table), a set of rules (rules produce new piece of information) and certain integrity constraints that have to be satisfied. One of the main advantages of deductive databases (in the past) was the ability to specify recursive rules. Furthermore, it was possible to view in recursive queries as well. In recent years, recursive queries have been implemented in different database management systems as well. Here, we list a few courses and their prerequisites (Fig. 2):

```
prereq(math, databases).
prereq(math, statistics).
prereq(databases, programming).
comes_before(X,Y):-prereq(X,Y).
comes_before(X,Y):-prereq(X,Z),comes_before(Z,Y).
```

Figure 2. Deductive database – an example

In [1], the idea of deductive data warehouses has been proposed. Things are similar to deductive databases but some important distinctions exist and they are explained in the paper. The paper also shows how to perform OLAP analysis on data, i.e., how to use Datalog rules to analyze data. In [8], the idea was extended in order to show how some other types of analysis could be implemented by means of Datalog rules as well.

In this paper, we show how Datalog rules can be used in order to reduce the size of a data warehouse. More precisely, we show how to implement implicit aggregate and derived tables. Although they are not physically implemented as such, their existence is sometimes effective to perform reasoning on data. Furthermore, we show that view materialization can significantly improve performances.

There are several different papers that explore the use of Datalog and its role in data warehousing. Boulicaut et al. [10], use rules in a similar way, but they focus on knowledge discovery. Neumayr et al. [11] use Datalog to reason over multidimensional ontologies. Aligon et al. [12] explore how to summarize and query logs of OLAP queries. However the term deductive data warehouses is new and papers on the topic cannot be found.

This paper is structured as follows: first the deductive data warehouse is described. Then we say a few words about aggregate tables, their role and ways how to implement them. In the next section, we say a few words about derived tables. In both sections we show how deductive rules can be used to specify derived and aggregate tables. Then we show some experimental results. Finally, the conclusion is presented.

II. DEDUCTIVE DATA WAREHOUSES

In this section, we say a few words regarding deductive data warehouses. Deductive data warehouses are quite similar to deductive databases, i.e., they both contain facts, rules and integrity constraints. When talking about deductive data warehouses, integrity constraints are not so important as in deductive databases because ETL designers are responsible that rules are obeyed and quality of data is ensured.

In [1], it was shown how Datalog rules can be used to simulate and perform data analysis including slice, drill down, what if, etc. The term used to describe the model was new. The idea seems to be important in the same way that deductive databases are important for data analysis in regular databases. Some rules (from [1]) are given below (Fig. 3). The first rule was used to find users. The second rule was used to perform drill down/roll up analysis on data. The third rule was used to perform what-if analysis on data. In [8], the idea was extended and some more complex rules were added to perform the Recency Frequency Monetary (RFM) analysis.

```
user(X):-users(_,X).

drill_down(X,Y,H,R):- X=year,
group_by((dates(B,F,Y,H,I,J), log(A,B,C,D)), [H],
R=sum(D)).

what_if(B,H,R,Z,W):-user_points(B,H,R,I,P),
A is I + Z, W is A * R.

compare(X,R):-addrange(Minv,Q1,Q2,Q3,Q4,Maxv),
((X >= Minv, X < Q1, R is 1);
(X >= Q1, X < Q2, R is 2);
(X >= Q2, X < Q3, R is 3);
(X >= Q3, X < Q4, R is 4);
(X >= Q4, R is 5)).
```

Figure 3. Datalog rules

In [1], a small example was used to show how deductive rules can be used to perform OLAP analysis on data. The scenario model was quite simple; users performed some actions on certain dates and we needed to analyze the number of actions (this is a part of a real project). For that purpose several tables were defined (Fig. 4):

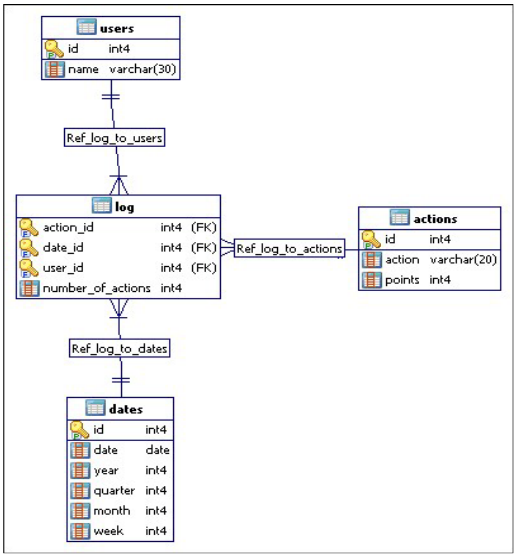

Figure 4. Data model

Some of the rules that were used to analyze data are listed above (Fig. 3). These rules were used to perform simple queries that are not so common in data warehouses, as well as some more complex queries. Some other rules were defined as well, but for more information we refer to [1] and [8].

In this paper we extend the idea and we show that Datalog rules can be used for other purposes as well, primarily for specifying aggregate and derived tables. In the next section, we show how to use rules to specify aggregate tables that are used to improve performances.

III. AGGREGATE TABLES

When we discuss tables that contain aggregate data, their purpose is quite simple. Since fact tables usually contain very large number of records, queries that use fact tables with large number of records can take too long. Although people that use data warehouses know that time needed to get results is much greater than the time needed when a query is posed against a transactional database, this doesn't mean that we are not interested in reducing that time. Since fact tables could contain millions of records, aggregation of a large number of records can last several minutes (or dozens of minutes). That is why we use tables with aggregated data, i.e., data that are pre-calculated and stored in order to improve query performances. By using tables with aggregated data queries can be answered much sooner, but on the other hand the size of a data warehouse grows (in addition to initial load of data and taking into account incremental loads that have to be carried out on a daily or weekly basis). But this is a trade-off that one does in order to speed up the access to relevant information.

Once aggregate tables are added into the system, data in aggregate tables have to be maintained as well. We do not discuss aggregate table maintenance any further (one can extract data from original sources or one can use tables from the data warehouse), but it is important that queries that are executed on aggregate tables return the same results as queries that use fact tables. Just to have in mind, when one uses aggregate and derived tables performances can be improved up to several hundreds or thousands time because the number of records (and I/O operations) becomes significantly reduced [9].

Once aggregate tables are created, the system needs to know how and when to use them. Although aggregate tables are created, that doesn't mean that it makes sense to use them always, but in some occasions (most certainly) they should be used. In order to use aggregate tables (when needed), there should exist a component that is called aggregate navigator [9]. It has a number of tasks, but the most important one is to know which aggregate table to use and when.

One Business Intelligence (BI) tool that is used in this paper is Business Objects XI. This tool has a construct called @Aggregate_Aware that is used to specify that aggregate table exists and that it should be used in certain queries (Fig. 5):

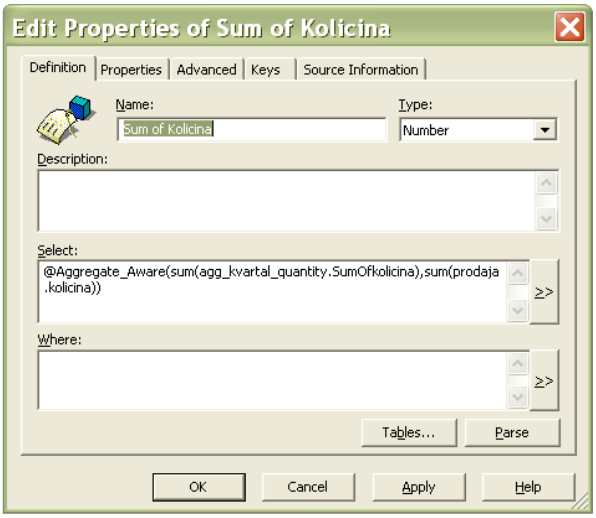

Figure 5. Business Objects XI - @Aggregate_Aware

When talking about aggregate tables, the most common scenario is to create such a table by means of a query that uses GROUP BY clause and SUM() as an aggregate function.

In this example, we show how to create a rule that (in fact) represents an aggregate table (log file analysis scenario), but it is not materialized (Fig. 6). Datalog Educational System (DES) was used to implement the rules and results are presented below the rules (the first row means that user 1, Smith Peter, committed 645 actions in January):

```
user_month_agg(A,B,K,R):-
group_by((users(A,B),log(C,D,A,F),dates(D,H,I,J,K,L)),
[A,B,K], R=sum(F)).

DES> user_month_agg(A,B,C,D).
{
user_month_agg(1,'Smith Peter',1,645),
user_month_agg(1,'Smith Peter',2,637),
user_month_agg(1,'Smith Peter',3,573),
user_month_agg(1,'Smith Peter',4,585),
user_month_agg(1,'Smith Peter',5,642),
user_month_agg(1,'Smith Peter',6,604),
user_month_agg(1,'Smith Peter',7,645),
user_month_agg(1,'Smith Peter',8,581),
user_month_agg(1,'Smith Peter',9,564),
user_month_agg(1,'Smith Peter',10,585),
user_month_agg(1,'Smith Peter',11,626),
user_month_agg(1,'Smith Peter',12,593)
}
Info: 12 tuples computed.
```

Figure 6. Data aggregation

Now, when we want to compute the sum of the number of actions on a quarter (year) level (hierarchy), the idea is that the system uses *user_month_agg* table and not the fact table any more. Namely, the original fact table (called log) could be transactional. This would mean that every action that user did in the past was stored in the fact table on a certain date and time. However, the log table that we have stores the number of actions for each day, so it is a periodic snapshot fact table.

 ISBN: 978-1-61208-346-9

In Business Objects XI it is quite easy to use hierarchies and such reports are easy to produce. In Designer tool one creates a hierarchy (we see Hierarchies Editor in Fig. 7):

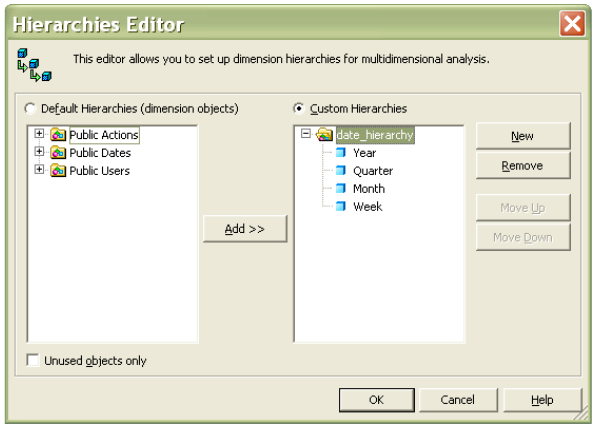

Figure 7. Business Objects XI - Hierarchies Editor

Once hierarchies are specified, one can use them in Desktop Intelligence tool. Mouse over quarter column offers drill down to a lower (Month) level (Fig. 8):

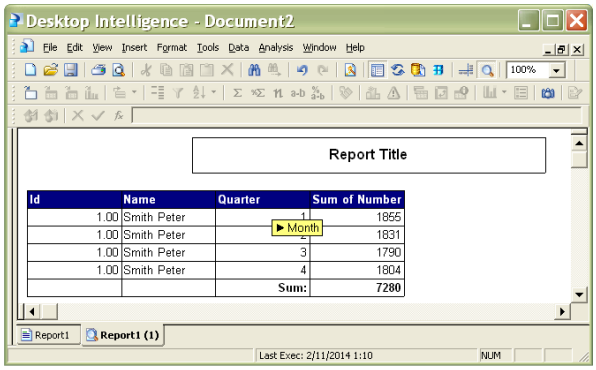

Figure 8. Number of actions (quarter level)

We can see the month level results (Fig. 9) that are only a few mouse clicks away:

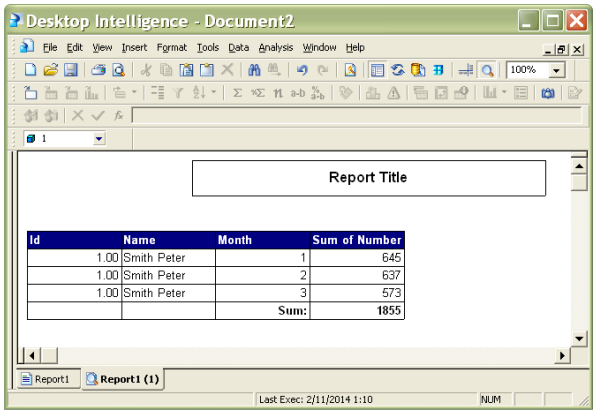

Figure 9. Number of actions (month level)

In order to calculate the number of actions on a quarter level by means of Datalog rules, we could define a rule (*uq*) that looks like this (Fig. 10):

```
uq(A,B,J,R):-
(group_by((users(A,B),log(C,D,A,F),dates(D,H,I,J,K,L)),
[A,B,J], R=sum(F))).

DES> uq(A,B,J,R)

{
uq(1,'Smith Peter',1,1855),
uq(1,'Smith Peter',2,1831),
uq(1,'Smith Peter',3,1790),
uq(1,'Smith Peter',4,1804)
}
Info: 4 tuples computed.
```

Figure 10. Number of actions – quarter level

The rule uses the log fact table and groups the records in order to get the result. The first row means that in the first quarter the user 1, Smith Peter, committed 1855 actions, etc. In DES, it took 17 seconds to answer the query. We can see that results in DES and in Business Objects XI are the same, as one could expect (Fig. 8 and Fig. 10).

But since we already have a rule that calculates the sum of actions on a month level, months could be easily aggregated to a higher (quarter) level. For a beginning let us add a rule that merges months and quarters (Fig. 11):

```
user_date(A,B,J,K,R):-
user_month_agg(A,B,K,R),
dates(E,F,G,J,K,I).

DES> user_date(A,B,C,D,E).
{
user_date(1,'Smith Peter',1,1,645),
user_date(1,'Smith Peter',1,2,637),
user_date(1,'Smith Peter',1,3,573),
user_date(1,'Smith Peter',2,4,585),
user_date(1,'Smith Peter',2,5,642),
user_date(1,'Smith Peter',2,6,604),
user_date(1,'Smith Peter',3,7,645),
user_date(1,'Smith Peter',3,8,581),
user_date(1,'Smith Peter',3,9,564),
user_date(1,'Smith Peter',4,10,585),
user_date(1,'Smith Peter',4,11,626),
user_date(1,'Smith Peter',4,12,593)
}
Info: 12 tuples computed.
```

Figure 11. Number of actions – month level

The first row means that the first user (Smith Peter) committed 645 actions in the first month of the first quarter, etc. Once months are joined with the date dimension, we can aggregate on other attributes using the date dimension (Fig. 12):

 ISBN: 978-1-61208-346-9

```
user_quarter(A,B,J,X):-
group_by(user_date(A,B,J,K,R), [A,B,J], X=sum(R)).

DES> user_quarter(A,B,C,D).
{
user_quarter(1,'Smith Peter',1,1855),
user_quarter(1,'Smith Peter',2,1831),
user_quarter(1,'Smith Peter',3,1790),
user_quarter(1,'Smith Peter',4,1804)
}
Info: 4 tuples computed.
```

Figure 12. Number of actions – user quarter

When called, DES needed 17 seconds to calculate the result (we restarted the program in between). So, implicit definition does not seem to be helpful except it just reduces the rule as such. Namely, this rule is used to give the same result (i.e., the number of actions on a quarter level), but it does not use the log fact table any more. It uses the implicit definition aggregate table, i.e., the *user_date* view. In the next section we try to see what could happen if the rule was materialized.

IV. MATERIALIZED VIEWS – EXPERIMENTAL RESULTS

However, if the view was materialized (physically), the time need to calculate the answer should be much smaller. The next SQL statement was used to materialize the view (Fig. 13):

```
SELECT u.id "user", u.name "name",
d.quarter "quarter", d.month "month",
SUM(l.number_of_actions)
INTO uqa
FROM users u INNER JOIN log l ON(u.id=l.user_id)
INNER JOIN dates d ON(l.date_id = d.id)
GROUP BY 1, 2, 3, 4
ORDER BY 3, 4;
```

Figure 13. View materialization (SELECT INTO statement)

This statement created a table called *uqa*. Once the view was materialized (in the form of a table), the results were calculated much sooner (it took less than a second). The next rule uses *uqa* table to perform the grouping and to calculate the result:

```
user_quarter_m(A,B,C,D,X):-
group_by(uqa(A,B,C,D,E), [A,B,C,D], X=sum(E)).
```

Figure 14. Using materialized view (uqa) in a rule

Of course, one has to have in mind that once data are aggregated to a higher level, some lower level queries cannot be answered any longer because the details are lost. More on materialized views can be found in [14].

Based on the previous discussion these would be some basic prerequisites that the aggregate navigator should possess. It should be capable to recognize that aggregate tables exist and it should be able to use them in situations when it makes sense. Here, we could extend the approach in order to make a Datalog aggregate navigator implementation but this could be done in our future papers. In the next section, we show how derived tables could be used.

V. DERIVED TABLES

In this section, we present different types of derived tables ([9]) and we show how to implement some of them in Datalog.

When talking about derived tables, several types can be distinguished. Pre-joined table is a table that consists of several tables that are joined together in order to speed up the querying. Further on, one can define derived tables that contain only a portion of data coming from original tables, etc. However, one has to have in mind that they may become quite big and they may require additional space.

The first example is used to demonstrate how Datalog rules can be used to create pre-joined tables. The rule name *dm* stands for *data mart* and it means that the rule would contain data from three different tables (users, log and dates), i.e., it would represent a complete data mart (only several rows are shown in the result):

```
dm(A,B,C,D,F,H,I,J,K,L):
users(A,B),
log(C,D,A,F),
dates(D,H,I,J,K,L).

dm(A,B,C,D,F,H,I,J,K,L).
…
dm(1,'Smith
Peter',4,362,1,date(2012,12,27),2012,4,12,52),
dm(1,'Smith
Peter',4,363,3,date(2012,12,28),2012,4,12,52),
dm(1,'Smith
Peter',4,364,6,date(2012,12,29),2012,4,12,52),
dm(1,'Smith
Peter',4,365,2,date(2012,12,30),2012,4,12,52),
dm(1,'Smith Peter',4,366,5,date(2012,12,31),2012,4,12,1)
}
Info: 1464 tuples computed.
```

Figure 15. Data mart specification

We can see that data from three different tables can be accessed easily, from a single (implicit) data mart. However, the time needed to calculate the data mart was a little less than 20 seconds.

Further on, we can define a rule that contains only a portion of data from the original fact table; one has to add a condition A=1 (Fig 16.):

```
portion(A,B,C,D):-log(A,B,C,D), A=1.

DES> portion(A,B,C,D).
…
portion(1,363,1,1),
portion(1,364,1,10),
portion(1,365,1,1),
portion(1,366,1,2)
}
Info: 366 tuples computed.
```

Figure 16. Partial fact table

Here we select only one portion of the fact table, more precisely only rows that refer to action_id = 1 (actions in the paper were events such as read, update and insert). There are other possible ways to produce derived tables (one can combine two of the already mentioned approaches) or define other derived tables. For example, one derived table could be used to transform data (certain measures) from original table if there was a need to do so, etc.

VI. CONCLUSION

In this paper, it has been presented how deductive data warehouses could use Datalog rules to specify aggregate and derived tables. On a number of examples it was shown how to use Datalog rules in order to explain how aggregate navigator should behave and to demonstrate how other types of derived tables could be built as well. A few reports were built in DES as well as in Business Objects XI and a small data warehouse was implemented in PostgreSQL database management system. Implicit table definitions may seem to be interesting as they do not require additional space, but only after view materialization we noticed that performances were improved significantly.

Based on the previous results ([1]) it is now clear that deductive data warehouses support OLAP analysis on data and some other (more complex) analysis as well. In this paper, we have shown how to add support for derived and aggregate tables and we showed that view materialization is good for data warehousing purposes.

In future papers one could look at how to implement aggregate navigator in Datalog. Furthermore, one can see that it is not practical to work with dimension tables that have large number of attributes. Because of that one could also explore and see how to create rules more easily and how to pose goals more intuitively.

REFERENCES

[1] K. Rabuzin, "Deductive data warehouses," IJDWM, in press.

[2] R. Kimball and J. Caserta, The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. Indianapolis, USA: Wiley Publishing, 2004.

[3] H. W. Inmon, Building the Data Warehouse – Third Edition. New York, USA: John Wiley & Sons, 2002.

[4] C. Ballard, D. Herreman, D. Schau, R. Bell, E. Kim, and A. Valencic, Data Modeling Techniques for Data Warehousing. [Online]. Available from: http://www.redbooks.ibm.com/redbooks/pdfs/sg242238.pdf. Retrieved on 16.04.2013.

[5] F. Silvers, Building and Maintaining a Data Warehouse. Boca Raton, USA: CRC Press, 2008.

[6] M. Novak and K. Rabuzin, "Prototype of a web ETL tool," unpublished.

[7] P. Ponniah, Data Warehousing Fundamentals: A Comprehensive Guide for IT Professionals. New York, USA: John Wiley & Sons, 2001.

[8] K. Rabuzin, A. Lovrencic, and M. Malekovic, "Using deductive data warehouses to analyze data", The Business Review, in press.

[9] C. Adamson, Mastering Data Warehouse Aggregates, Solutions for Star Schema Performance, USA: Wiley Publishing, 2006.

[10] J. F. Boulicaut, P. Marcel, and C. Rigotti, "Query driven knowledge discovery via OLAP manipulations", [Online]. Available from: http://liris.cnrs.fr/~jboulica/bda01.pdf. Retrieved on 30.10.2013.

[11] B. Neumayr, S. Anderlik, and M. Schrefl, „Towards ontology-based OLAP: datalog-based reasoning over multidimensional ontologies", DOLAP '12 Proceedings of the fifteenth international workshop on Data warehousing and OLAP, 2012, pp. 41-48.

[12] J. Aligon, P. Marcel, and E. Negre, "Summarizing and querying logs of OLAP queries", Advances in Knowledge Discovery and Management, 471, pp. 99-124, 2013.

[13] H. C. Tjioe and D. Taniar, "Mining Association Rules in Data Warehouses", IJDWM, vol. 1, pp. 28-62, 2005.

[14] J. V. Harrison and S. W. Dietrich, "Maintenance of materialized views in a deductive database: an update propagation approach", [Online]. Available from: http://www.public.asu.edu/~dietrich/publications/MaintenanceOfMaterializedViews.pdf. Retrieved on 25.04.2013.

 ISBN: 978-1-61208-346-9