The Logical Model and Language Support for Patterns in DBMS

Zdeňka Telnarová, Martin Žáček

***Abstract*—**The authors' aim is the Logical Model and Language Support for Patterns in DBMS (Database Management System) and builds on the previous work of the authors. Patterns are mentioned usually in the extraction context. Little stress is posed in their representation and management. This paper is focused on representation of the patterns, manipulation with patterns and query patterns. Crucial issue can be seen in systematic approach to pattern management and specific pattern query language which takes into consideration semantics of patterns. Example of implementation of the abstract pattern model is proposed. Query language for manipulating with patterns is introduced on Time Series data.

***Keywords*—**Pattern type, Association Rule, Cluster, Time Series, Query Language.

I. Introduction

GROWING possibilities of fast and cheap data collection and its storing in large databases go hand in hand with requirements for effective manipulation and interpretation of such data. Ways of data storing, models which provide bases for such ways, are the defining factors for effective manipulation and information collection, knowledge respectively. There are increasing requirements to capture behavior of objects in a database, to capture data semantics, or mutual relations between data and possibilities of knowledge inference from data stored in databases. The crucial factor is the ability of abstraction, i.e. modelling at such a level of abstractions so that the model could cover as many particular situations as possible, possibly being universal. An interesting idea is to store knowledge in a form of behavior patterns. The question is: Can a behavior pattern be written universally? Can we create a model that would offer such a high level of abstraction? An interesting answer for this question can be found in [1], which presents a model of behavior patterns as a basis for a layer between raw data and the language for data interpretation. This approach distinguishes between models of data and models of patterns [2].

This work was supported by the University of Ostrava grant SGS10/PřF/2017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of the sponsors.

Z. Telnarová is with the Department of Informatics and Computers, Faculty of Science, University of Ostrava. 30. dubna 22, Ostrava, the Czech Republic. (e-mail: Zdenka.Telnarova@osu.cz).

M. Žáček is with the Department of Informatics and Computers, Faculty of Science, University of Ostrava. 30. dubna 22, Ostrava, the Czech Republic. (e-mail: Martin.Zacek@osu.cz).

The paper follows up formal definitions and tries to outline implementation of a general model aiming at effective storing while using various techniques of recognized or user-defined patterns in a database.

Pattern-base system can be characterized according to [3] by the following properties:

1) Abstraction. Patterns providing the user with a meaningful abstraction of raw data.
2) Efficiency. By specific model system improves the efficiency of both traditional transactions and advanced processing on patterns.
3) Flexible querying. Query language manipulating with patterns has to be modified in terms of improvement possibility to retrieve and compare patterns.

II. ABSTRACT PATTERN TYPE

Definition

A patter type PT is a quintuple [N, SS, D, MS, MF]: where N is unique identifier of the pattern type; SS is Structure Scheme - distinct complex type; D is Domain - s a set type; MS is Measure Schema – a tuple of atomic types; MF is a Mapping Formula - a predicate over SS and D.

Each patter is instance of a specific patter type. Structure schema depends on specific pattern type as is shown later for several specific pattern types. There is different structure for association rule, cluster, time series, etc. Definition below comes from the same quoted work.

Definition

A patter p over a pattern type PT is a quintuple [PID, S, AD, M, MF]: where PID is a unique identifier S is Structure (instance of Structure Scheme); AD is the Active Domain, a relation which instantiates the set type of Domain; M is Measure – valid values of the respective structure and measure schema of PT; MF is mapping formula – predicate instantiating the respective mapping formula of PT

As we mentioned Structure Scheme depends on specific patter type. It occurs useful to define pattern class as a collection of semantically related patterns which are instances of specific pattern type. For example we can have pattern class AssociationRule as collection of all instances of the specific pattern type Association Rule.

Definition

A pattern class is a triplet [Name, PT, Extension]: where name is a unique identifier of the class;PT is a pattern type; Extension is a finite set of the patterns with pattern type PT.

III. Specific Pattern Types

Though we consider pattern types as a general concept some of concrete types depending of pattern recognition algorithm can be shown. Let us mention only a few of them like Association Rule, Interpolating Line, Time Series, cluster, etc.

A. Pattern type Association Rule

N: AssociationRule
SS:TUPLE (head: SET(STRING), body: SET(STRING))
D: BAG (transaction: SET(STRING))
MS: TUPLE (confidence: REAL, support: REAL)
MF: $\forall$ x (x$\in$ head $\vee$ x$\in$ body $\Rightarrow$ x $\in$ transaction)

Confidence describes what percentage of the transactions including the head also include the body. Support describes what percentage of the whole set of transactions include the body.

Instance of the Pattern AssociationRule

PID: 1
S: (head = {‘Programing with PL/SQL’},
body= {‘Databases’, ‘Programing with SQL’})
AD: ‘Select name AS transaction FROM subjects ‘
M: (confidence = 0.75, support = 0.55)
MF: {transaction: $\forall$ x (x$\in$ {‘Programing with PL/SQL‘}
$\vee$ (x$\in$ { ‘Databases’, ‘Programing with PL/SQL’}
$\Rightarrow$ x $\in$ transaction)}

This pattern describes situation when student that assigns for Databases and Programing with SQL also assigns Programing with PL/SQL with confidence 0.75 and support 0.55.

Merit for this approach can be seen in the fact that many algorithms and techniques how to mind association rules have been developed. This technique has many practical applications. Patterns instance is easy to devote in SQL.

B. Pattern type Cluster

N: Cluster (two dimensional)
SS:TUPLE (center: TUPLE (x: REAL, y: REAL),
radius(r: REAL))
D: BAG (a: INTEGER, b: INTEGER)
MS: precision: REAL
MF: (BAG.a - TUPLE.center.x)2 +
+ (BAG.b - TUPLE.center.y)2 <= TUPLE.radius.r2

Instances of the Pattern Cluster

Let us have customers and know their age and incomes. Via cluster analysis we have obtained clusters, each cluster contains customer with similar age and income. For example we can obtain these clusters:

Cluster 1		*Cluster 2*	
age	*income*	*age*	*income*
30	*33*	*43*	*60*
31	*31*	*47*	*60*
29	*29*	*45*	*59*
30	*27*	*49*	*61*

The actual patterns for data from BAG can be described this way.

PID: Cluster1
S: (TUPLE(center(x: 30, y: 30), radius(r: 3))
AD: ‘Select age, income FROM customers ‘
M: Precision: 1
MF: (age - 30)2 + (income - 30)2 <=32

PID: Cluster2
S: (TUPLE(center(x: 45, y: 60), radius(r: 2))
AD: ‘Select age, income FROM customers ‘
MF: Precision: 0.75
MF: (age - 45)2 + (income - 60)2 <=22

This approach benefits from the techniques based on N-dimensional representation of data. The source of data is a set of N-dimensional elements. It depends on the characteristics of the data whether this approach is profitable to use. If we have data as a set of N-dimensional elements then the merit is obvious.

IV. Implementation of the abstract model in Oracle

To be able to store patterns in pattern base it is necessary to create a structure of the pattern base. Pattern base can be modeled by simple meta-meta model shown on FIGURE 1. Patter_type is a general structure - entity which instances are specific types of patterns. Specific types can be added to pattern base according to the situation when some other specific type occurs in reality. This new instance of pattern_type creates specification for particular pattern it means concrete structure of attributes and their domains. For example Association Rule can be instance of patter_type and this specification concretizes the structure of the entity pattern.

Pattern_type entity is abstract entity that is specified only by definition of attributes and description of their meanings through informal text specification. Pattern entity (as a instance of pattern_type entity) has formal declaration according to concrete specification belongs to specific pattern_type. FIGURE 2 shows example of data type model for abstract data type in Oracle for patter_type Association Rule.

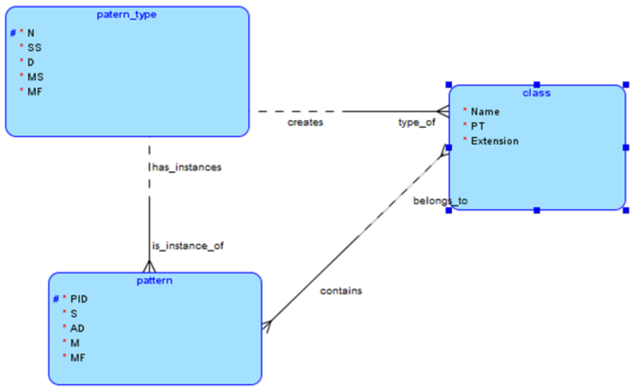

Fig. 1 meta-meta model for pattern.

N: AssociationRule
SS:TUPLE (head: SET(STRING), body: SET(STRING))
D: BAG (transaction: SET(STRING))
MS: TUPLE (confidence: REAL, support: REAL)
MF: $\forall$ x (x$\in$ head $\vee$ x$\in$ body $\Rightarrow$ x $\in$ transaction)

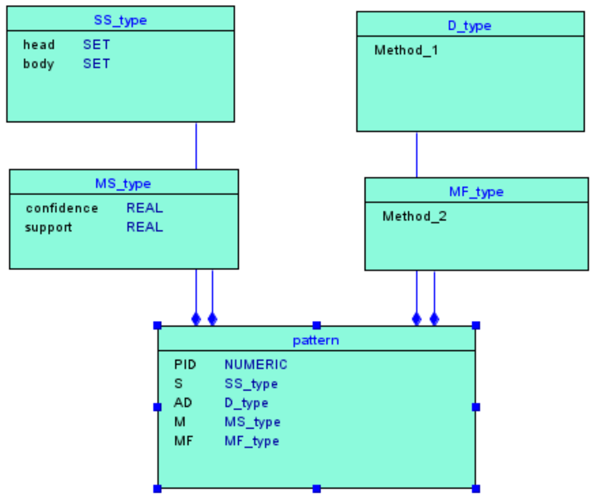

Fig. 2 Model of abstract data type in Oracle.

Method_1 is implementation of selecting data from specific database, Method_2 is implementation of mapping formula (predicate over SS and D).

Figure 3 is an example of data model for generation of schema for storing patterns. This model is based on Oracle abstract data type collention (set of strings for head and body) and object (Method_1 and Method_2).

V. *Pattern Type Time Series*

A time series is a set of timestamped data entries. A time series allows a natural association of data collected over intervals of time. For example, summaries of stock market trading or banking transactions are typically collected daily, and are naturally modelled with time series [4].

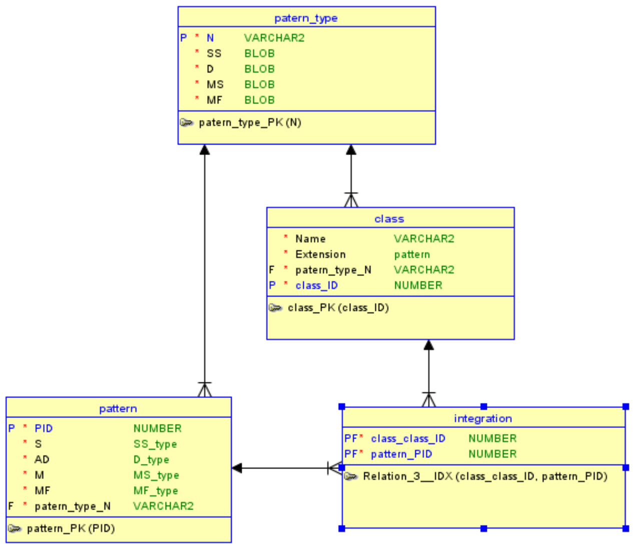

Fig. 3 Data model uses Oracle abstract data types.

A time series can be regular or irregular, depending on whether or not the time series has an associated calendar.

1) A regular time series has an associated calendar. In a regular time series, data arrives predictably at predefined intervals.
2) An irregular time series does not have an associated calendar. Often, irregular time series are data-driven, where unpredictable bursts of data arrive at unspecified points in time or most timestamps cannot be characterized by a repeating pattern.

Data generation for a time series begins with individual transactions. Each transaction has a timestamp and sufficient information to identify that transaction uniquely. Individual transaction data is typically *rolled up* to produce summary data for a meaningful time period.

Using of this approach depends of the characteristics of the data. This approach assumes data with timestamp. If we have timestamped data entries pattern type time series is the most useful, so the merits are evident. Many applications work with time series for example when we work with historical data.

A. Time Series as Historical Data

1) The data-collection model for historical data has the following characteristics:
2) At daily intervals, historical data is updated with daily summary data (main update cycle).
3) At some period after the main update cycle, corrections of the daily summary data may need to be applied.
4) Queries may be executed at any time, even during the update cycle.
5) Queries do not observe the current day's summary information until after the main update cycle has completed.

This historical data is modelled using multiple regular time series.

Example of historical data

An organization needs to keep data about employees' salaries. All employees are paid weekly. Initially, the following EMPLOYEE entity was modelled.

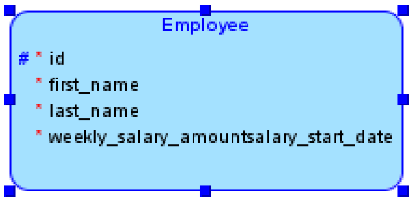

Fig. 4 Employee table.

Additional requirements now specify that the organization needs to keep a historical record of how and when employees' salaries have changed during their employment.

To model salary changes over time, add a SALARY HISTORY entity.

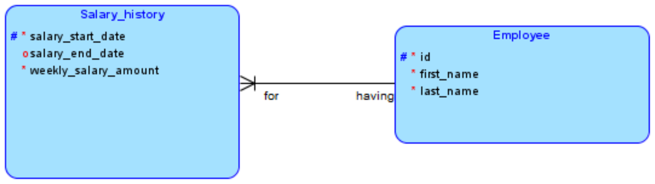

Fig. 5 Decomposition to the historical structure for weekly salary.

The UID of the Salary_history entity is the related Employee id and the salary_start_date.

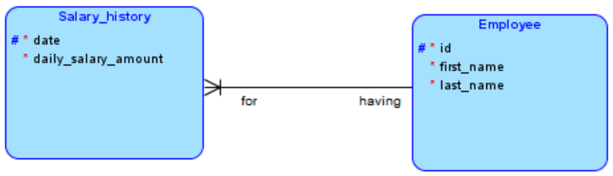

Fig. 6 Decomposition to the historical structure for daily salary.

The UID of the Salary_history entity is the related Employee id and date.

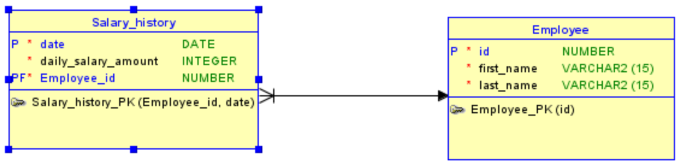
Salary_history
P * date DATE
* daily_salary_amount INTEGER
PF* Employee_id NUMBER
Salary_history_PK (Employee_id, date)
Employee
P * id NUMBER
* first_name VARCHAR2 (15)
* last_name VARCHAR2 (15)
Employee_PK (id)

Fig. 7 Decomposition to the historical structure for daily salary.

SQL DDL statements

```
CREATE TABLE Employee
 ( id        NUMBER NOT NULL ,
   first_name VARCHAR2 (15) NOT NULL ,
   last_name  VARCHAR2 (15) NOT NULL  );
ALTER TABLE Employee ADD CONSTRAINT
Employee_PK PRIMARY KEY ( id );

CREATE TABLE Salary_history
 ( date          DATE NOT NULL ,
   daily_salary_amount INTEGER NOT NULL ,
   Employee_id       NUMBER NOT NULL  );
ALTER TABLE Salary_history ADD CONSTRAINT
Salary_history_PK PRIMARY KEY ( Employee_id, date ) ;
ALTER TABLE Salary_history ADD CONSTRAINT
Salary_history_Employee_FK FOREIGN KEY ( Employee_id
) REFERENCES Employee ( id ) ;
```

B. *Typical features of Time Series DBMS*

Time Series DBMS are designed to efficiently collect, store and query time series data with high transaction volumes. Although that type of data could be managed with other categories of DBMS (and some systems even provide appropriate design patterns or even extensions for handling time series), the specific challenges often benefit from specialized systems by supporting:

Downsampling data: e.g. a sensors value is stored per second and a query shall deliver the averaged value per minute. A typical SQL-query needs a group by clause with an expression similar to something like 'group by integer division(time, 60)', whereas Time Series DBMS support something like 'group by time (1 minute)'

Comparison with the previous record: e.g. a 'table' contains stock prices (one tick per day). A query should deliver all days, in which the price of a specific stock had increased. In a relational system (and using standard SQL), we would have to self-join the table and figure out how to match each record with its previous one. That may be a non-trivial task and is definitely not efficient. Time Series DBMS typically offer specific features for that requirement.

Joining time series: Joins will put two or more time series together, by matching timestamps. Those timestamps, however, may not match exactly. Time Series DBMS often provide features for that task.

For the one or other system, we found it difficult to decide whether to add it as another Time Series DBMS, or to classify it as a monitoring application (which we do not take care of). We then applied as a rule of thumb, that for being a DBMS, a system at least has to offer an API for inserting and querying data and must not be specific to a single domain.

Oracle Time Series provides support for time series domain-specific types, functions, and interfaces. The product focuses on a set of time series data representation and access mechanisms sufficient to support many applications and the development of more specialized time series functions. The objects option makes Oracle (from the version Oracle8i) an object-relational database management system, which means that users can define additional kinds of data - specifying both the structure of the data and the ways of operating on it - and use these types within the relational model. This approach adds value to the data stored in a database.

Informix Time Series software consolidates and organizes time-stamped data much more efficiently than traditional, relational databases and provides the following benefits:

More efficient storage of data means that Informix Time Series performance is an order of magnitude greater than that of traditional, relational databases. Additionally, a real-time loader is included that dramatically reduces data load times to make data available to queries in real-time.

The inherent ability to store time series data with great efficiency means Informix Time Series requires significantly less storage space than traditional relational databases. Informix Time Series often uses one third the storage space that is required by a standard relational database.

Informix Time Series enables consistent, scalable performance that helps yield highly predictable costs. Benchmarks have illustrated that Informix Time Series keeps storage requirements linear over time.

High availability. Informix can also replicate Time Series data throughout a distributed, clustered environment.

The Time Series capability of Informix is “built-in” to the server; no additional installations or licenses are required.

VI. LANGUAGE SUPPORT FOR PATTERN

The need to look up patterns in database is common for many applications. Many applications require work with sequential data. Query languages in databases are strong tool for obtaining information from data. The question is how to accommodate them to be able to handle complex queries on sequences. There are two different approaches to query patterns. It is based on the form of mapping formula:

1) predicate form of mapping formula,
2) mapping formula is expressed by procedure/function or method of abstract data type (ADT).

If the mapping formula is a method of ADT or procedure/function then evaluating formula means call the method. In case mapping formula is a predicate we deal with the declarative representation of the formula and module that computes the formula by logic program is needed.

Tasks for query language:

Pattern matching – if new pattern is loaded into set of patterns the question is whether this pattern is already found in the storage. Another question is what data corresponds to new loaded pattern.

Deduce new pattern based on existing – patterns can be composed with part-of relationship.

Meta querying – query deals not only with patterns but also with pattern type. This idea is very new. It was mentioned in [5] and introduced binary pattern operators.

Although most commercial Data Base Management Systems support extensions to provide a library functions that can be called from a SQL query there is still lack of expressive power, flexibility and integration with database query languages. Some systems for improving this situation were proposed like SQL extension called SEQUIN [6] for querying sequences or extension of the relational algebra with sequence operators for sorted relations called SRQL. Another quite interesting approach can be seen in SQL-TS language which adds to SQL constructs for specifying sequential patterns.

VII. SQL-TS LANGUAGE

According to the [7] SQL-TS for TimeSeries adds to SQL simple constructors for sequential patterns [8]-[9] like CLUSTER BY and SEQUENCE BY. A CLUSTER BY clause specifies what data is processes separately. A SEQUENCE BY clause specifies according what attribute the data must be traversed by ascending. Definition of the pattern is a part of the FROM clause in SELECT statement. The AS clause is used to specify a sequence of tuple variables from the specified table.
The statement is similar to common SELECT statement with GROUP BY and ORDER BY clauses.

Example of the SELECT with CLUSTERED BY and SEQUENCE BY clauses:

The statement selects employees that daily_salary_amount went up by 5% or more one day and then down by 10% or more the next day.

```
SELECT X.id
FROM Salary_history
   CLUSTER BY id
   SEQUENCR BY date
   AS (X, Y, Z)
WHERE Y.daily_salary_amount > 1.05*
X.daily_salary_amount
   AND Z. daily_salary_amount<0.9*Y. daily_salary_amount
```

In the figure 7 we can see the effect of SEQUENCE BY and CLUSTER BY statement.

Tab. 1. the data is group by Emplyee_id and in each group is ordered by date

Employee_id	daily_salary_amount	date
…	…	…
100	200	1/1/15
100	205	1/2/15
100	190	1/3/15
…		
101	300	1/1/15
101	290	1/2/15
101	340	1/3/15
…		

X, Y, Z in the statement represents [10] three tuples that immediately follow each other. Tuple variables can be used in WHERE clause for definition of the conditions and also in SELECT clause for definition of output. Using tuple variables we avoid creation of complex SQL query that requires three joins. Another benefit from this way of querying sequential data is much easier way to optimize the query. In general a star denotes a sequence of one or more tuples that satisfy all conditions in WHERE clause.

Example of the SELECT expressing recurring patterns by using a star operator

The star operator is used to specify a sequence in which the value of specified attribute tends to decrease or increase according to specified condition.[11]

```
SELECT X.Employee_id, X.date AS start_date,
Z.previous.date as end_date
FROM Salary_history
CLUSTER BY Employee_id
SEQUENCE BY date
AS (X, *Z, Z)
WHERE Y.daily_salary_amount < Y.previous.
daily_salary_amount
AND Z. previous. daily_salary_amount <
0.5*X.daily_salary_amount
```

VIII. Conclusion

The article focuses on the logical model and language support for patterns. In the chapter I. the motivation of using abstract pattern types is presented. In the chapter II. needed definitions of the pattern types and patterns as instances of the pattern types are introduced. Specific pattern types association rules and clusters are described in chapter III. Chapter IV. focuses on the implementation issues. For the implementation Oracle SQL was used. Chapter V. gives its thought to time series pattern type and to its specific occurrence historical data. The main attention of the article is devoted to time series patterns and time series as historical data are discussed. Implementation in Oracle is attached. Chapter VI. And chapter VII concern with language support for patterns. In terms of language support SQL-TS language is presented and there is an example of select statement to illustrate CLUSTERED BY and SEQUENCE BY clauses.

The idea how to store , manipulate and query patterns is not very new but still important and there is many tasks that is necessary to solve. Analogue to the storing, manipulating and querying relations or object classis is obvious but the way is not equal. Patterns have specific properties and characteristics that are necessary to consider. There two approaches how to manipulate with patters. The solution is either to store raw data separately to patterns or to have common storage space. In both cases pattern base management system for managing patters is needed. One of the solution was introduced by PSYCHO [12]. This approach deals with Pattern Bases Management System (PBMS) engine where pattern base consists of pattern types (meta data), patterns (data), class definitions (meta data) and instances of classes (data). PBMS has these modules: pattern definition language, pattern modification language, pattern query language. In future we would like to pay attention on pattern language based on relational calculus and relational algebra in more detail. The main requirement on the pattern language is to preserve generality of the pattern model without reference to particular pattern type.

References

[1] S Rizzi, E. Bertino, B. Catania, M. Golfarelli, M. Halkidi, M. Terrovitis, P. Vasiliadis, M. Vazirgiannis and E. Vrachos: “Towards a Language for Patters”, In Proc. of the 22nd International Conference on Conceptual Modeling (ER 2003), Chicago, 2003.

[2] Z. Telnarova, J. Schenk, “The logical model for pattern representation”. In: International Conference of Numerical Analysis and Applied Mathematics 2015, ICNAAM 2015, p. 120009. DOI: 10.1063/1.4951892.

[3] I. Bartolli et al., “Patterns for next-generation database systems: preliminary results of the Panda project”. In Proc. 11th SEBD, Cetrano, Italy, 2003.

[4] M. Žáček, “Introduction to time series”, In Pattern Recognition and Classification in Time Series Data (pp. 32-52). IGI Global. 2017.

[5] 5. M. Terrovitis, P. Vassiliadis: Architecture for Pattern-Base Management Systems, online, http://citeseerx.ist.psu.edu/viewdoc/similar?doi=10.1.1.60.8704&type=sc

[6] P. Seshadri, M. Livny, R. Ramakrishnan: Sequence query processing. In Proceedings of ACM SIGMOD Conference on Management of Data, 1994

[7] R.Sadri, C. Yaniolo, A. Yarkesh, J. Adibi: Optimization of Sequence Queries in Database Systems, Online http://web.cs.ucla.edu/~zaniolo/papers/pods2001.pdf

[8] Miarka, R., & Žáček, M. (2011). “Knowledge patterns for conversion of sentences in natural language into RDF graph language”. Paper presented at the *2011 Federated Conference on Computer Science and Information Systems, FedCSIS 2011,* 63-68.

[9] R. Miarka, R., M. Žáček, “Representation of knowledge patterns in RDF(S) „. Paper presented at the Proceedings of the 13th International Conference WWW/Internet 2014, ICWI 2014, 147-154. 2014.

[10] Z. Telnarova, „Data modelling and ontological semantics“. International Journal of Data Analysis Techniques and Strategies, 4(3), 237-255. doi:10.1504/IJDATS.2012.047818.2012.

[11] Z. Telnarova. „Modeling and language support for the pattern management“. Pattern recognition and classification in time series data (pp. 86-106) doi:10.4018/978-1-5225-0565-5.ch004. 2016.

[12] B. Catania, A. Madalena, M. Mazza : (A Prototype System for Pattern Management

Birth: Vaclavovice, 29.12.1955. Education: 1996-2001 Czech Republic, Technical University of Ostrava, Faculty of Economics, Systems Engineering and Informatics Ph.D.; 1992 – 1993 Czech Republic, University of Ostrava, Pedagogical Faculty, Pedagogical specialisation; 1975 – 1980 Czech Republic, Technical University of Ostrava, Faculty of Economics, System Engineering and Informatics Master degree.
She has been working for University of Ostrava, Czech Republic, Faculty of Science, Assistant professor, Department of Informatics and Computers. Before she worked for Technical Secondary school as a teacher and for Kancelarske stroje (Office machine) as mathematician, analyst.