COMPLEX MANAGEMENT IN TEMPORAL DATABASES

Michal Kvet, Karol Matiaško

Žilinská univerzita v Žiline, Faculty of Management Science and Informatics
e-mails: Michal.Kvet@fri.uniza.sk, Karol.Matiasko@fri.uniza.sk
Slovakia

Abstract: Databases and data management form the core part of the current processing in the information systems. Data tuples are mostly delimited by the validity time frame. Nowadays, it is strictly necessary to store not only current valid data, but the whole evolution. Therefore, the temporal paradigm has been created, which is based on the object identifier extension. Thus, the primary key of the object is extended by the time definition. This paper offers the cross-section of the temporality management, historical evolution and proposes new data models characterized by the various granularity of the processing. Our own proposed attribute oriented temporal model is based on storing any granularity and time spectrum, as well. Whereas some data columns are not necessary to be monitored over time, even some of them do not evolve at all, the mentioned solution brings a suitable power. In the paper, we deal also with architecture improvements, which can manage synchronization groups. The first part of the paper deals with the architecture of temporality. The second part evolves from the data management, background processes up to volatility and identification of the version. The third part proposes the transaction definition and the impacts of integrity.

Key words: temporal database, integrity, attribute, synchronization group, transaction.

1. INTRODUCTION

One of the main and most significant parameters expressing the quality and usability of the information system and technology itself is related to the data approach and data access effectivity. In the past, the total amount of managed data was low, data were either a direct part of the application or stored in the file system with no universal data structure [5].

Nowadays, there is a significant pressure to deal with the complex data, to manage not only current valid data but the whole evolution, as well [6, 11].

Relational paradigm is not, however, prepared to model time delimited data in a complex manner. It is characterized by the conventionality property [5]. Thus, if the data object tuple is updated, the original version is replaced and the system does not store the original value anymore. Data management, analytics and robust decision making is, however, based mostly on the historical data with emphasis on the future prognoses, as well. Therefore, it is necessary to bring the solution operating data in any valid time sphere.

The aim of this paper is to propose an overview of the temporal database architectures with emphasis on the various characteristics and properties. The first part of the paper deals with the temporal architecture evolution. Section 2 covers historical solutions and state of the art. In section 3, temporal architectures are defined. Two of them were proposed in 2015 and 2017 by our research. Universal solution integrating existing functionality is defined by the synchronization groups. Thanks to that, there is no problem with change identification and anomalies. The second part of the paper is covered by the temporal extension of the approaches and technologies – transactions, integrity, and volatility.

2. HISTORICAL EVOLUTION

The task for data management has been originated soon after the first releases of the information systems. It can be even said, that it is correlated with the development of the information technology and hardware themselves. During the first phases, data were an inseparable part of the systems, they were simply embedded and could not be shared anywhere else. Later, researcher and developer's task was to split them forming data files, which were mostly maintained as sequential [1, 2]. First database systems were created in the 60ties of 20th century. Relational paradigm is a core part for the data management, which is based on relational theory, mathematical algebra and relational correlations. Data are formed in the shape of tables and relationships between them. This theory is now widespread and most often used [1, 6].

Limitation of the system is just the data in the time spectrum. Data tuples characterize only current valid states, although the data tuple can be bordered by the validity. In this case, however, it does not reflect temporality, whereas there is no possibility to store individual versions [4].

First attempts of historical databases were recognized with the arrival of transaction theory. Each data change is automatically stored in the transaction log to ensure the consistency and durability of the whole database. It means, that any change is stored in the log file system before the change itself. Thanks to that, historical images can be calculated from the UNDO and REDO image [9]. Although such a concept can deal with history, physical implementation and real usage are far more complicated, even impossible. First of all, log files are cyclically rewritten, therefore the historical images are very time delimited. In the real environment, it can be reflected even in the second, minute up to hour frame [10].

Later, to ensure complex integrity, no possibility for data loss and database corruption, the archive log mode principle was proposed. In that case, each log file is copied to the archive repository before its cleaning. Thus, any image can be produced from history, if all the log files are accessible. The significant task is, however, based on the efficiency of the whole process. To reconstruct an image from the history, current (non-archived) log files must be scanned and evaluated. Afterward, archived log files are applied, up to the time definition. Strict limitation of such an approach is covered by the fact, that the log file does not contain information about the objects, which were maintained by it. So, all of them must be searched and evaluated. Principle of the image reconstruction is shown in fig. 1.

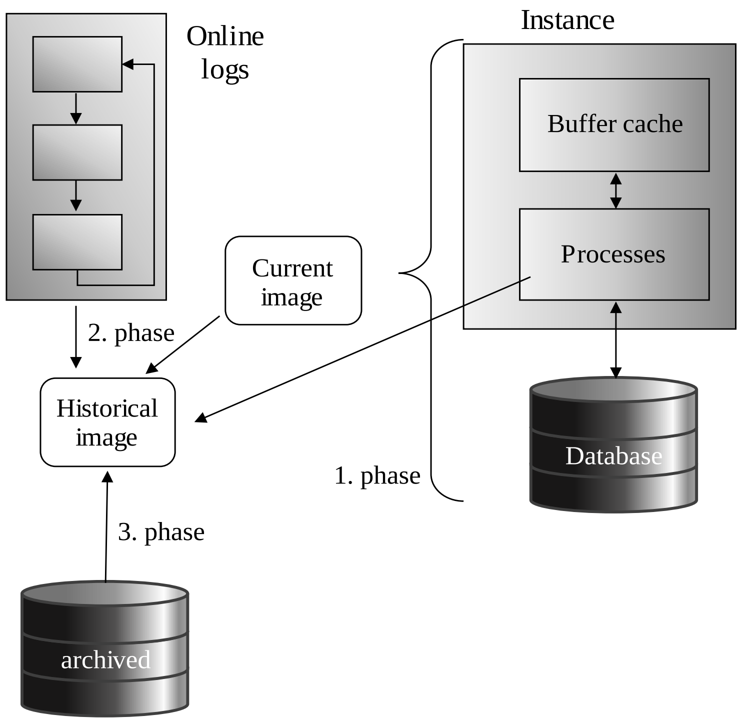

Fig. 1. Historical image composition

In 80ties of the 20th century, log file management regarding historical images was improved. The header of the log file was extended by the list of objects, which changes are stored inside. This was, however, applied only as of the extension of the archived log file system. Evaluation and calculation of the header were produced during the archiving process and supervised by the master *Archiver* background database process [9, 10]. But there is a performance problem - the log file could be overwritten just after it was archived. Such activity was the bottleneck of the system. Another approach from this category works differently – if the online log file is full, it is copied to the archive repository twice. The first version is used for building historical images. Inside the second copy, the header is constructed. It is necessary to store two versions during the process of the header creation because of the file non-availability for other database processes. After the processing itself, the first copy is removed, access and evaluation are done according to the header. Limitation of such an approach is the fact, that two copies must be created, although they are stored only in a time-limited manner. Thanks to the header definition, performance can be improved up to 80% - regarding the number of transactions and versions of the particular object [8]. Construction of the historical image is defined by these phases: in the first phase, the current image is obtained directly from the database. Afterward, online log files are applied (second phase). In the third phase, archived log files with no created header are scanned completely in a sequence. The third phase is more effective, whereas the header of the archived log file is scanned. If there is no evidence of the change of the particular requested object, the whole file processing is skipped. Vice versa, if there is any notice in the header, the file is scanned completely in a sequential manner. The last step consists of the image construction from the particular data obtained from the log system. In 90ties of 20th century, strong research pressure to optimize the performance of the historical image construction could be felt. Although the header of the log file improved performance, there was still the necessity to scan the file, if there is any notice about the object. There was no searching possibility, which weakened the system. Thus, the aim was to propose a solution for searching objects inside the log file. Whereas there were attempts to index data inside the database, there was an open question to apply such rules and techniques to the log system, as well. The result of the research was designed soon forming two streams. The first is based storing index access structure directly in the header of the file [4, 12]. It meant, that after archiving, the particular data file has to be marked as unavailable during the index construction. As already described, this resulted in a necessity to store two versions in a limited time. The second approach is characterized by the index, which is stored directly in the database [10]. Archived log file is transformed to be similar to a data file of the database in a logical structure. The main advantage is based on removing the requirement to store two versions of the same file.

Mentioned approaches are able to deal with historical images, respectively are able to construct them on request. On the other hand, the process can be significant time and resource-demanding, if there is a requirement to get the high data amount, e.g. image of the whole database or table at a defined time point, even interval or to monitor the evolution. Such a problem is caused mostly by the log file principle, which is not optimal for the temporality. The log itself consists of the UNDO and REDO image of the row [13, 14]. During the processing, it must be checked, whether the transaction has been even approved, whereas also data of rejected transactions are inside. Moreover, there are also description data of the transactions and system themselves. Thus, there is also information, which is useless for temporal evolution management. Finally, technology does not produce the whole temporal system. It can deal only with the historical data. Future valid plans cannot be managed, at all. Moreover, archived data log is external from the database point of view, thus the database does not have information if some of the files are removed, lost or corrupted. As a consequence, it is not ensured, that the produced result data are consistent and reliable [15].

Even more sophisticated solutions based on *Flashback technology* do not solve the problem complexly. Although they offer effective data management of history, whereas they are based on improved log file performance, it is not possible to deal with future plans. *Flashback technology* produces own files, based on current log images. These files consist only of the data relevant for the temporal evolution – e.g. attributes values, which are not changed, are not stored inside [1, 3].

Concluding this section, we can divide the data with regards on temporality to the following categories [8]:

- conventional – data attribute values can be changed, but there is no necessity to monitor them over the time,
- temporal – temporal evolution must be handled,
- static – values cannot be changed later, at all (e.g. code lists).

3. TEMPORAL ARCHITECTURES

Soon after approving relational paradigm based on the conventionality, researchers and developers perceived the consecutive necessity for dealing with temporal data effectively. Although in that time, hardware and technology were expensive consequencing in strong limitation for temporal data management, the aim was to propose at least models, concepts and principal approaches. Fig. 2 shows the first temporal model, which is even used many times even in these days. It does not propose a new temporal paradigm, only the extension of the conventional technology is used. Identifier of the object in a relational form is defined by the primary key characterized by two properties – uniqueness and minimality (any sub-element of the primary key would lose the property of uniqueness) [3, 4]. By using such a set of attributes, it is possible to identify the row exactly. To observe that concept, the primary key was extended with the time element. Thus, the object itself can be identified by several rows of non-overlapping time frames. Object-level temporal model is shown in fig. 2. The first category is conventional with no opportunity to monitor changes over time. The second model is the uni-temporal. The primary key of the object is a composite consisting of the identifier of the object (*ID*), begin point of the time frame (*BD*) and end point of the time frame (*ED*). It can be considered as a uni-temporal model. Time interval mostly characterizes validity. The bi-temporal solution consists of two-time spheres. Its aim is to cover also state corrections. The third model in fig. 2 is bi-temporal. In that case, the primary key consists of at least five attributes. In general, the multi-temporal solution can be used. It is necessary to mention, that the interval can be modeled using various representations – closed or open from the left and right side [5, 6]. The specific solution is defined by only one attribute. In this case, each newer state automatically delimits the direct predecessor.

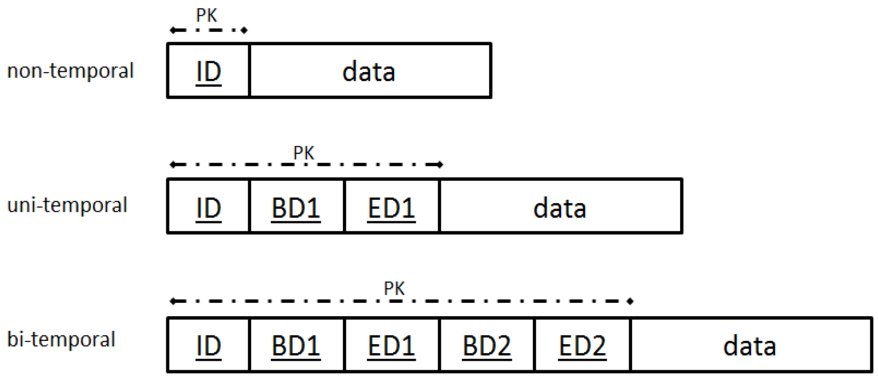

Fig. 2. Object-oriented temporal model

State of the object can be even undefined or invalid, either partially or completely. Such a situation must be carefully modeled and covered to ensure the correctness, robustness, and reliability of the whole solution [8, 11, 14].

Object-level granularity approach proposes an optimal solution if the changes of the attributes are synchronized and each operation updates all of the attributes in the object definition. It is, however, many times too strict limitation. Therefore, it is necessary to create also another solution, which cannot be too deeply derived from the conventional paradigm of the relational database.

The second model was defined by our research and is characterized by attribute granularity. The architecture consists of three layers. The first level contains only current valid data in the conventional form. Thanks to that, existing applications based on only current valid data can work without any change in the database connection and structure. The second level forms the core part of the temporality and consists of the temporal table referencing any temporal change. Each attribute definition is extended by the definition of the temporal access, whether individual changes are to be monitored or not, respectively other temporal conditions. This layer is internal, operated by added background processes [8, 9]. The third level stores non-current valid data – historical data and future plans. It is operated by only background processes. External systems have only read-only privileges to the second and third level. The architecture of the solution is defined in fig. 3. Notice, that this solution divides individual changes to the attribute granularity. Each temporal change of the attribute is stored in the temporal level by adding a new row. Thus, if only one attribute value is changed, all other attributes are not changed at all with no reflection to the temporality. As a consequence, there are no duplicates. Thus, from such point of view, performance and architecture are optimized. Unfortunately, there is one significant performance limitation defined by the synchronizations. If several attributes are updated at the same time, all of them are processed separately consequencing in creating a bottleneck of the system in the second (temporal) level of the structure.

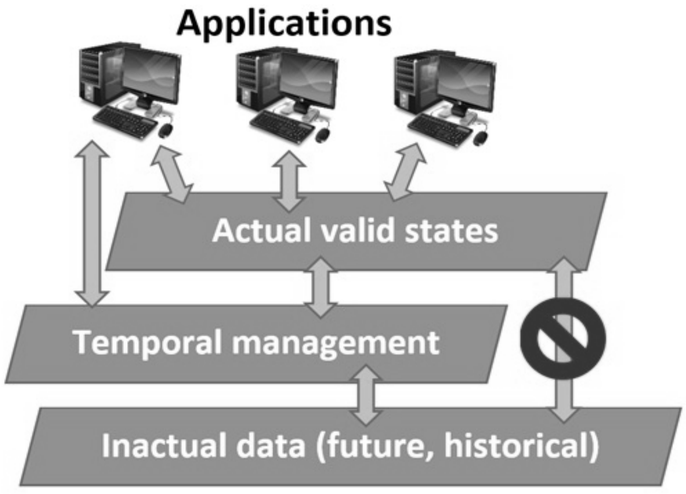

Fig. 3. Attribute oriented temporal model

In 2017, we proposed a temporal system, which aim is to cover the space between attribute and object approach. Our solution is named temporal group architecture. Its origin is based on attribute oriented system, but it leaves its granularity and composes attribute groups, which are maintained as one attribute. In this case, the system itself does not monitor the attribute itself, but the synchronization groups, which can even consist of only one attribute. Thus, it can cover all the systems from the object up to attribute granularity. Group management and detection are evaluated automatically based on the input data update stream. The architecture contains six levels, the extended core part is defined by three layers detecting, managing and constructing synchronization groups. Fig. 4 shows the architecture. Physical view of the group is shown in fig. 5. Group can be directly constructed from attributes, but as the combination of the groups or group and attribute, as well [7, 12]. Each created group is temporally delimited by the validity frame. During the evolution, groups can be created, altered and dropped dynamically, based on the synchronization processes outside the system forming the update stream. Processes ensuring group management are delimited by the parameters expressing a number of consecutive synchronized updates to create a new group. Another parameter rates the unsynced changes to reconsider designed groups. The attribute can be part of multiple groups simultaneously. In this case, however, it is necessary to ensure, that those update events cannot occur at the same time, whereas there would be a collision.

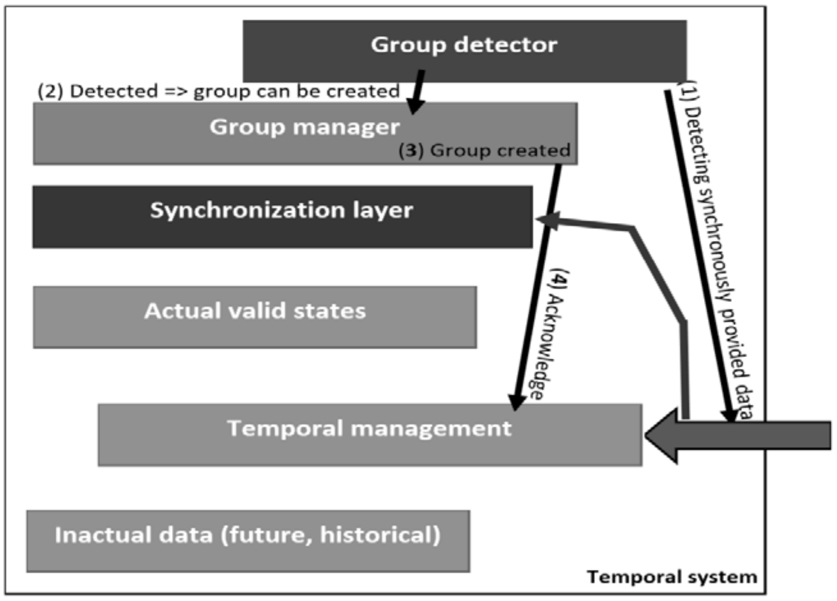

Fig. 4. Group-oriented temporal model

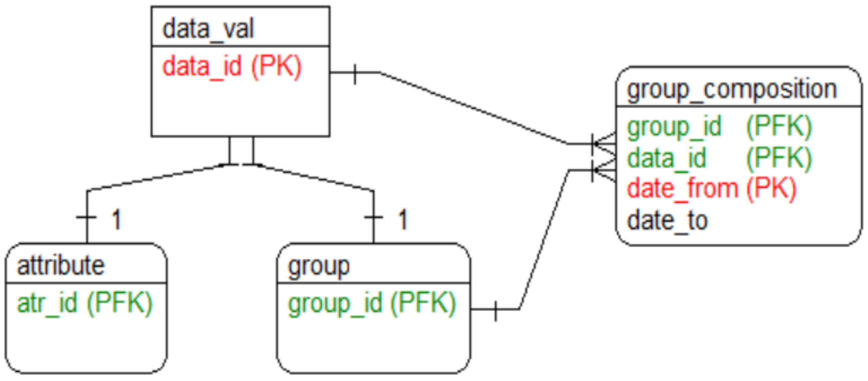

Fig. 5. Temporal data group management

The following sections of this paper propose techniques for transaction management with an emphasis on the integrity of the whole system.

4. TRANSACTIONS

Relational databases are characterized by strong demand of the transaction processing. Any change, even its attempt is covered by the transaction, which ensures the consistency and durability of the processed data, even after the crash.

The fundamental of the processing is just the log management, each action is stored in the log file located in the physical storage consisting of the original and new image of the tuple. Thanks to that, it is always possible to request commit and rollback of the transaction. Moreover, if there is a requirement of other transaction to get an image of the object, using log files, such data can be produced. In the field of the temporality, a transaction has many more functions to ensure processed data to be correct, to ensure all requirements and rules are passed. In conventional systems, management is protected by the two-phase transaction protocol – the first phase is characterized by the writing process to the log followed by the second stage – change application to the physical database. If the transaction is not approved, a particular request for the change is noticed only in the log, with no implementation in the database.

For the temporal environment, I now propose a new three-phase protocol. The principle is similar to the existing system, but the temporal evaluation module in the database must be added. In the first stage, data are written to the log file. If there is at least one temporal attribute to be processed, the system acknowledges the temporal background processes, which ensure, that the data are stored in the temporal module (in attribute or group granularity architecture – it reflects temporal layer). For now, particular data are marked as not approved. Such data in the temporal layer are always committed. Thus, after the second stage, complex data have not been approved, whereas the main transaction has not been ended, yet. Now, the system waits for the user to end the transaction. It must be either aborted (rollbacked) or committed. These factors are operated in the third stage. If there is a requirement to commit data, all of them are administered in the main database. In the temporal layer, the flag of particular rows is changed to approved state. Vice versa, if the transaction is to be refused, data are not stored in the main database, just the transaction log is closed for the transaction. Temporal layer marks relevant rows to get state of the refuse. Model of the three-phase transaction processing is shown in fig. 6.

The integrity of the temporal system arizes from the conventional paradigm. It can be divided into these five sections – *column* (*NULL, NOT NULL*, unique characteristics), *user*, *referential*, *entity* (object identification by the temporal primary key), *domain* (data type of the attribute with reference to temporal evaluation and monitoring). The core part of the temporality is just the user and referential integrity. Commonly, data model consists of several tables, which are interconnected using relationships based on the primary and foreign key. A foreign key can hold either undefined value (in a case on the non-mandatory type) or candidate of the primary key of an appropriate table. If the object granularity is used, it is necessary to check also time spectrum covering [5] [7]. It can, therefore, happen, that one object has to reference multiple instances of the master table.

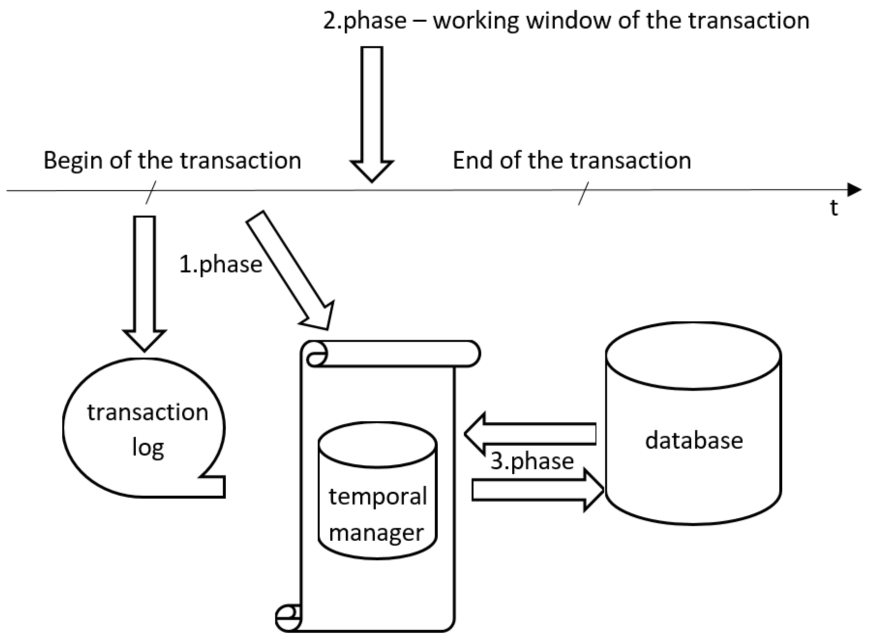

Fig. 6. Three-phase transaction protocol

The problem is shown in fig. 7. Object *B* is defined during the time interval <T5; T6) and references object *A*. Object *A* is composed of three partial states (*S1, S2,* and *S3*) delimited by the following time intervals:

- S1 <T1, T2)
- S2 <T2, T3)
- S3 <T3, T4)

Let assume these rules:

- T1 < T2 < T3 <T4
- T1 < T5 < T2
- T3 < T6 < T4

If the object *B* must be covered by the interval defined by object *A*, three approaches can be used – *strict, one* and *ignore*. The *strict* rule requires a referential object to cover all the object in the master table. Therefore, object *B* must hold all three states of object *A* – *S1, S2,* and *S3*. References are commonly stored in the nested of the mapping table. The second approach is based on the rule „*one*“. Object *B*, in that case, references only one state, which is at least partially covering. Usually, the first suitable from the left side is selected. If the whole state is not covered, consecutive states are then selected. The problem occurs if there are time intervals, during which states are undefined or there is a problem with the reliability of such states. The last option is *ignore*. It does not check the time interval and covering of individual states, at all.

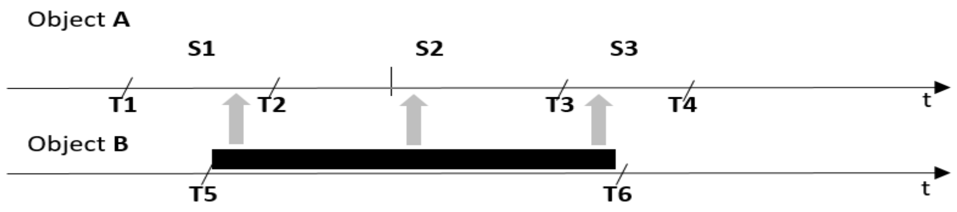

Fig. 7. Object referencing

5. VOLATILITY

Although the aim of the temporal approach is to store time-limited states over time, many times, too historical data do not provide any additional benefit to the system. A significant problem can also arise as a part of the user integrity, where states can be stored in the system only limited time or passing the specific conditions. GDPR is very strict for the data management and necessity to remove such state or the whole object can often occur. It is, therefore, necessary to propose modules for dealing with the data to ensure consistency. Based on our long term research, we propose five methods, which cover such states (they can be completely removed or moved to another repository – aggregation, external, data warehouses, etc.):

- Purge_count_local
- Purge_count_global
- Purge_time_BD
- Purge_time_ED
- Direct

Data to be removed are specified by the time interval, during they can occur in the system based on the start (*Purge_time_BD*) or end point (*Purge_time_ED*) of the validity or by the number of consecutive changes (*Purge_count_local*) or states (*Purge_count_global*) of such object. Notice, that change can be even defined by only one attribute, whereas complex state expresses the change of all attributes. The direct approach reflects external rules – particular data are removed immediately based on the occurrence of the event outside the database. More about the volatility rules and techniques can be found in [8, 9].

6. CLASSIFICATION

Temporal approaches and architectures do not have specific categorization and classification consequencing in the difficult and demanding description of the data management and structures. In 2015, basic classification was proposed [8] forming three levels:

- Database system type (N – no DBS support, R – relational, X – object-relational, O – object, A – non – relational, U – unspecified).

- Temporal structure (N – non-temporal, U – uni-temporal, B – bi-temporal, T – three-temporal, M(x) – multi-temporal, where x expresses the number of temporal spheres).
- Transaction processing (N – nontransactional, L – OLTP with logs, O – OLTP with temporal objects, A – OLTP with attribute granularity).

Temporal evolution has brought the necessity to extend such principles and to cover the complexity and access technology more properly. In this paper, therefore, we propose more classification rules consisting of these parts:

- Database system type
- Temporal dimensions – description of the data with emphasis on the temporality (static, temporal, hybrid). Temporality is modeled based on the physical architecture – local or global point.
- Processed granularity for the database, table, object and attribute, as well.
- Data representation in a timeline – historical, current, future. This category also covers the significance of the change rule.
- Transaction processing.
- Collision management – principles of solving conflicts in the object state definitions, whereas no more than one state can be defined for any object during any time point.
- Index definitions and approaches with emphasis on its localization in the system (e.g. separate tablespace definition and its parameters). This category manages also data access methods. In 2018, the new technique limiting the necessity of the whole table scanning was proposed as the result of our research. Such method is named *Flower Index Approach (FIA)* and it is based on the principle of using index only for the data located inside the physical *database* files removing the impact of the data fragmentation after the Update statement processing and applying the aspect of the volatility.

7. CONCLUSIONS

Data management in a temporal manner brings significant element in the complexity of consecutive evaluation, analytics, decision making and creating future prognoses. As evident from the paper, it is not only about adding time elements to the processing, but the whole concept and approach must be changed to preserve the performance of the whole system. In this paper, three temporal architectures are defined. Group data level architecture defines a universal solution accepting any data structure and time aspect category. Hybridity is ensured by three layers managing synchronization groups.

Temporal data orientation and the consistency of the system is ensured by the transactions, which require a transition from the conventional paradigm to temporal, thus each data change, regardless its consecutive approval, is registered in the autonomous temporal storage. The aim is to provide a consistent image during any defined time frame (interval or just one-time point). Too historical values, however, do not provide benefit and such data needs to be removed from the main structure, alternatively moved to the specific archive repository. In this paper, an overview of our defined techniques of volatility is described.

The main contribution of this paper is to describe and reference the temporality, to point out its importance, architecture, approaches and, last but not least, the classification rules.

ACKNOWLEDGMENT

This publication is the result of the project implementation:

Centre of excellence for systems and services of intelligent transport II., ITMS 26220120050 supported by the Research & Development Operational Programme funded by the ERDF.

The work is also supported by the project VEGA *1/0089/19 Data analysis methods and decisions support tools for service systems supporting electric vehicles* and *Grant system* of the University of Zilina.

"PODPORUJEME VÝSKUMNÉ AKTIVITY NA SLOVENSKU
PROJEKT JE SPOLUFINANCOVANÝ ZO ZDROJOV EÚ

REFERENCES

[1] Ashdown, L., Kyte T. Oracle database concepts, *Oracle Press*, 2015.

[2] Avilés, G., et al. Spatio-temporal modeling of financial maps from a joint multidimensional scaling-geostatistical perspective. *Expert Systems with Applications*, pp. 280-293. 2016

[3] Doroudian, M., et al. Multilayered database intrusion detection system for detecting malicious behaviours in big data transaction, *IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, 2016

[4] Erlandsson, M., et al. Spatial and temporal variations of base cation release from chemical weathering a hisscope scale. *Chemical Geology 441*, pp. 1-13. 2016

[5] Johnston, T. Bi-temporal data – Theory and Practice. *Morgan Kaufmann*, 2014.

[6] Johnston, T., Weis, R. Managing Time in Relational Databases, *Morgan Kaufmann*. 2010

[7] Kvet, M., Matiaško, K., Temporal Data Group Management, *IEEE conference IDT 5.7. – 7.7.2017*. pp. 218-226. 2017

[8] Kvet, M., Matiaško, K. Transaction Management in Temporal System, *IEEE conference CISTI 18.6. – 21.6.2014.* pp. 868-873. 2014

[9] Kvet, M., Matiaško, K. Uni-temporal modelling extension at the object vs. attribute level, *IEEE conference UKSim, 20.11 – 22. 11.2014*, pp. 6-11. 2014

[10] Kuhn, D., Alapati, S., Padfield, B. Expert Oracle Indexing Access Paths, *Apress*. 2016

[11] Kumar, N. Efficient data deduplication for big data storage systems, *Advances in Intelligent Systems and Computing 714*, pp. 351-371. 2019

[12] Li, S., Qin, Z., Song, H. A Temporal-Spatial Method for Group Detection, Locating and Tracking, *In IEEE Access, 4,* 2016.

[13] Mehmood, R., Graham, G., Big Data Logistics: A health-care Transport Capacity Sharing Model, *Procedia Computer Science Volume 64*, pp. 1107-1114. 2015

[14] Ramírez, M., Moreno H., Millán N. Big Data and Health "Clinical Records", *Innovation in Medicine and Healthcare 2017*, pp. 12-18, 2017

[15] Rusnák, P., Rabčan, J., Kvaššay, M., Levashenko, V. Time-dependent reliability analysis based on structure function and logic differential calculus. *Proceedings of the thirteenth international conference on dependability and complex systems DepCoS-RELCOMEX*, 2019.

Information about the authors:

Michal Kvet – researcher and teacher in the Faculty of Management Science and Informatics, University of Zilina. His primary research area is directed to the databases, relational and non-relational form, optimization, architecture and performance. Temporal databases and approaches cover his long term research by implementing new techniques in the field of intelligent transport systems and hospital technology and information systems.

Karol Matiasko – vice-rector of the University of Zilina associated to the complex information systems, their interoperability and interconnections. His research strategies are related to the database systems in the relational, text and object form. Results of his research are implemented in many systems and presented to the students during lectures.

Manuscript received on 07 August 2019