Temporal database management

Temporal registration

Marek Kvet, Michal Kvet
Faculty of Management Science and Informatics
University of Zilina
Zilina, Slovakia
{Marek.Kvet; Michal.Kvet }@fri.uniza.sk

***Abstract*—Temporal databases are characterized by significant data amount monitored over the time. Such values are stored with regards on validity during the whole time spectrum supporting decision making, reliability definition, analysis, progress monitoring and creating future prognoses. When dealing with various granulairty of changes, management must be shifted into attribute granularity level to remove duplicates. In this paper, we define our own proposed temporal architecture with regards on element registration in the system. Whereas data model, defined attributes as well as temporality definition evolve over the time, it is necessary to create complex environment and possibilities for dealing with these changes and reflections to the temporal management layer. I tis done by temporal registration concept.**

Keywords- element registration; temporal approach; attribute oriented approach; temporal evolution;

I. Introduction

Temporal databases are based on assumption of storing significant data amount with different types over the time. In such systems, many data granularity types are used with various precision and frequency of changes. Typically, data retrieval is composed as image of the whole system or its sub-part at some time point or monitors individual changes of the elements over the time. Therefore, storage effectivity is strongly important, when dealing with data characteristics and evolution over the time. Advantage of conventional approach storing only actual valid data is just hte fact, that each new data tuple replaces existing one with no reflection to the history. On the one hand, we are losing history and evolution, however great advantage is just the simplicity, if the structure must be changed, either by adding, removing individual attributes, changing their data types or if the structure should be changed more complexly. After executing *Alter,* respectively *Create* or *Drop* commands, each change is directly and immediately applicable. In temporal environment, situation is more complicated, because also evolution of the data model must be highlighted, defined and stored, to create possibility to reconstruct not only states, but also the structure (data model) itself. In case of changing precision – increasing accuracy, such problem is not so sharp, because it can be done automatically at the cost of ineffective data management in the past – values are stored with high precision, but the data themselves do not have such accuracy. In this paper, we define individual possibilities of changing data characteristics, definitions and model itself over the time with regards on the temporal managemennt, data registration and reflection over the time.

In the first part of the paper, brief history introdution is proposed focusing on current temporal technologies. Afterwards, temporal table definition and classification is proposed with regards on the pure attribute oriented architecture developed by us. Individual characteristics and definitions can be extended by various principles based on group management [17], transaction processing [15], decision making, tuning [19], indexing [12] [18], reliability [13] and distributed data modelling [1] [20]. Nowadays, strong stream of temporal definition extended by the positional, data forming spatio-temporal solution can be perceived. Principles are defined in [4] [5] [11] using spatio-temporal characteristics.

II. Historical evolution and elements

Requirement for temporal data definition has become strong in the earlier part of the database systems proposal themselves. In the first phase, security and high accessibility were provided by the backups and transaction log files. By accessing such structures, historical image of the data could be obtained. It was, however, too complicated process due to necessity of backup loading, which required user intervention. Moreover, such backup usually covered the whole database or its major part, therefore it lasted too much time, mostly if data were located externally in separate storage capacity. Operational data for decision making could not be obtained at all. If the log file was deleted, there was no evidence about that fact, because database system itself does not manage historical (obsolete) log files, whereas it was used only for security reasons, if some physical file was lost or corrupted. As the consequence, it could not be used complexly and could even provide insufficient data without any user warning. Fig. 1 shows the principles. Let have two backups managed and stored by *RMAN* (*Recovery manager).* Log files themselves, however, cannot be supervised by *RMAN*, thus any lost of its part cannot be detected. As the consequence, from the log file loss period until complete backup is obtained, particular states can be incomplete and therefore considered as unreliable. However, user does not have any evidence about such problem, therefore he cannot not react any way [2] [3].

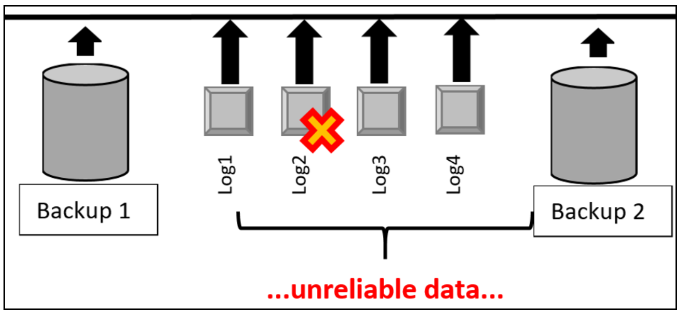


Fig. 1. Backup and log file data loss problem

Therefore, later, such functionalities and principles have been extended by the temporal transaction log definition and supported by the *Flashback technology* [3] [6] [9]. Concept has been proposed in Oracle 10g version and improved in Oracle 11g version. It allows you to define own methodology, own format and associate individual attributes to be managed temporally. It has brought the reduction of the data stored – only temporal attributes are monitored without any transaction information stored directly. Thus, data structure and evidence is optimized. It is automatically managed by *Flashback Repository process* based on defined attribute associations. Individual files are linked, referenced and managed automatically, so there is no possibility to delete referenced file. Thanks to that, provided solution is reliable, robust and powerful, provided data are correct. On the other hand, effectivity is ensured, only if the whole or partial data image should be obtained. In that case, it works like backup, but can be provided with regards on any timepoint in the past. Retrieving attribute or group evolution over the time is really complicated process resulting in inapplicability of defined solution, whereas it would require getting all partial images and changes reconstruction. Time processing of such data retrieval requirement is comparable with pure backup and log file management. Another limitation is just the system dependency – it can be used only in DBS Oracle. Other systems do not support it.

Fig. 2 shows the principles of *Flashback repository* management. If the data are updated and the change reflects temporal attribute, *Flashback Repository process* becomes active and stores particular historical data into *Flashback repository.* Thus, if the *Select* statement requires only actual data, they can be provided directly, historical data are accessed only by concacting *Flashback Repository process.* Data retrieval is then composition of the data provided by *PMON* (*Process Monitor*) – actual data, historical data are loaded from *Flashback repository* by particular process. Future valid data as core part of the temporality cannot be handled at all. Performance of such solution in comparison with our proposed solution is in section 6.

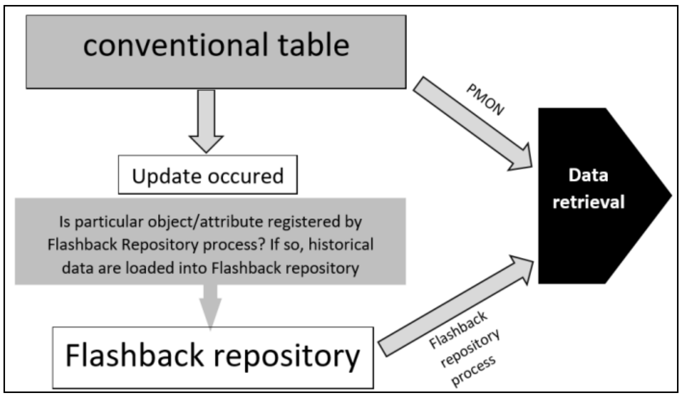

conventional table
Update occured
Is particular object/attribute registered by Flashback Repository process? If so, historical data are loaded into Flashback repository
Flashback repository
PMON
Data retrieval
Flashback repository process

Fig. 2. Flashback repository management

The second research and development stream is based on temporal definition using the direct schema data. Thus, primary key of the table considered as temporal is extended by the time definition. In principle, it reflects at least validity, however, in real environment, such primary key extension can deal also with transaction definition (transaction validity), position, time occurrence, etc – fig. 3. However, there are several disadvantages to be mentioned. First of all, it requires data updates to be synchronized – if not all attributes are updated at the same time, several duplicate tuples would be stored. Moreover, such table must be fully temporal – each attribute is temporal regardless its definition. Last, but not least problem would cause table joining together – child validity intervals must be fully covered by master (parent) states (fig. 4). If it cannot be done by one tuple, composition must be executed to check correctness. In fig. 4 – there are 3 parent records associated with the same object. Each state is delimited by the time validity (*T1, T2, T3*). When joining with another table, particular child state must be covered by two parent records (*S1, S2*). If undefined object state would be defined between individual intervals, *join* would not be possible to execute at all, because of the covering [7] [8] [10].

As we can see, such developed solution is not user friendly, correct and robust management is too complicated. Therefore, the aim of the researchers was to create new temporal paradigm and management automatically covering illustrated problems. Validity intervals were transformed into values of the *period* data type proposed in temporal paradigm. Complete definition has been proposed even at the end of the 20th century with the aim to standardize it. Unfortunately, such definition has not been approved and was recommended to be improved. After several standardization attempts and consequent rejection of each one, defined concept has stopped to be developed and abolished completely in 2001. To conclude such management definition, it is available, but must be handled explicitly [2] [3].

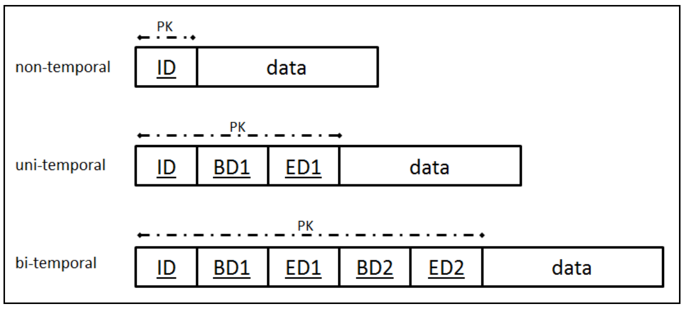


Fig. 3. Object level temporal definition [9]

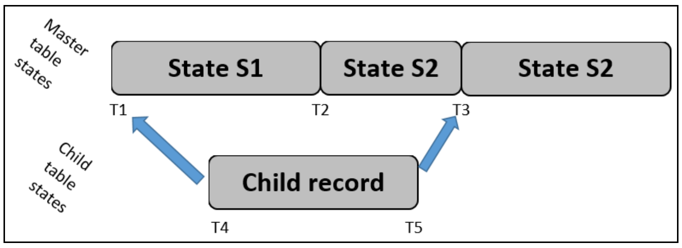


Fig. 4. States covering (referential integrity)

III. Own solution

In the previous chapter, research streams have been proposed with regards on the application usage and limitations. Our own proposed solution uses the benefits of the both stream systems with emphasis to eliminate boundaries. In comparison with validity temporal solution (it uses object granularity), our proposed solution uses attribute as the main element. Thanks to that, if granularity and frequency of the changes is not the same, no duplicate values are stored. It is based on three layers – the first layer deals with current valid states connected to the temporal layer, which forms the core of the system. Each change on temporal attribute is automatically referenced by such layer to ensure possibility of the state compositions. Non-actual states are stored in the third layer. It shows the perspective of the historical data as well as states valid in the future, which are automatically transformed to actual at defined timepoint. Architecture in shown in fig. 5.

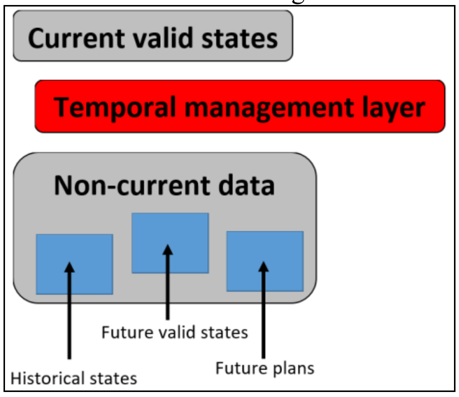


Fig. 5. Attribute oriented solution architecture

This concept has been developed by us and proposed in 2015. Data management, principles, properties can be found in [15] [16]. Later, the solution has been extended and optimized. In the pure way, each change of the temporal attribute directly invokes new *Insert* statement execution in the temporal layer. And it may be the source of inefficiency. Almost every temporal system is delimited by the sensor data processing (e.g. patient monitoring, industrial environment, transport systems). In principle, provided data have different granularity, precision and error handling. On the other hand, many times, to ensure data and measurement security, multiple sensors are used and consequently compared to create error immune measurement solution. In that case, partial data synchronization should be mentioned. Forming synchronized data into temporal groups provide effective solution and reflects performing only one statement into temporal layer. If the data are desynchronized, group is dropped or reconstructed either automatically or based on user requests. Principles are defined in [17].

Proposed attribute temporal approach with its improvements provide sufficient power, when dealing with data with various types, definitions, granularity and frequency of changes. It can robustly react to any change of the values.

In principle, each data attribute or tuple can be characterized by its temporal sphere. We can distinguish the following categories:

- ***Static*** – values stored inside cannot be changed later at all (onitialization parameters, code lists).
- ***Conventional*** - non-actual data are not necessary to be stored.
- ***Temporal*** – each attrribute definition is monitored over the time.
- ***Hybrid*** – combination of the previous options.

From this point of view, such defined attribute approach is hybrid.

So far, all temporal solutions have only dealt with the possibility of changing data, but the structure itself has remained the same. Temporal characteristics evolve over the time, provided data quality is still better and better, reliability and precision is higher. Therefore, also data model and complex structure must react to those situations and propose management of such changes with regards on structure defined in the past, but also states with impoved data model.

Therefore, in this paper, we deal with the possibilites to manage and solve such problems. Architecture of the proposed solution must be extended, specifically, another layer managing temporal model and structure is proposed. Processing is related, evaluated and secured by registration layer, which is managed by introduced background process – *Temporal Structure Manager (TSM)*. It is the master process and if the number of changes is high, also executor processes can be defined (either automatically or manual) – *Temporal Registration Executor Processes (TRegExecn)*.

IV. Logical view

Temporal *Data definition language* (*DDL*) statements are defined as the extension of the existing syntax by using kedwords definiting *temporality*. Access model can be defined either using object or attribute granularity. If object granularity is used, each attribute is influenced by the definition. We propose the following temporality keyword options based on time management:

- ***Static*** (attribute value cannot be changed at all), applicable for table as well as attribute itself.
- ***Conventional*** (attribute can be changed at any time, historical value is not necessary to be stored later at all), applicable for table as well as attribute itself.
- ***Temporal*** (attribute is monitored over the time, each change is stored and accessible over the time), applicable for table as well as attribute itself,
- ***Hybrid*** (special type of the table, which allows you to store any type of the previously mentioned characteristics in the common table). Applicable only for the table. It cannot be used for attributes at all.

Fig. 6 shows the syntax of table definition command. In this case, keyword characterizing temporality is placed before the table structure and its name:

```
Create [{static | conventional | temporal | hybrid}] table table_name
 (atr1 data_type1,
  atr2 data_type2,
     …
 );
```

Fig. 6. Table with temporality definition

Vice versa, *temporality* keyword can be defined for the attribute. The syntax of the solution would look like following (fig. 7). The first attribute (*atr1*) is *conventional*, the second attribute (*atr2*) is *temporal* and the last third attribute (*atr3*) is *static*.

```
Create table table_name
 (atr1 data_type1 conventional,
  atr2 data_type2 temporal,
  atr3 data_type3 static
 );
```

Fig. 7. Temporality of individual attribute definition

The previously defined commands are defined for table creation. Similar principle is also used for altering structure:

Fig. 8 reflects the transformation to the temporal, fig. 9 shows the syntax of individual attribute (*atr1*) transformation.

```
Alter table table_name convert to temporal;
```

Fig. 8. Changing existing table temporality

```
Alter table table_name convert attribute atr1 to temporal;
```

Fig. 9. Changing existing table attribute temporality

Object (table) as well as attribute dropping is a bit more complicated process. In that case, it is necessary to distinguish, whether defined object to be dropped is at least partially temporal (*temporal, hybrid*) or not (*conventional, static*). If data were not monitored over the time, particular object can be dropped directly. However, if the definition reflects temporality, two situations can occur:

1. particular referenced object can be dropped – in that case, keyword *purge* is used meaning that the whole reference is removed (from historical tables, planned calendar, future plans, actual states and also temporal table). Information about the existence in the past, however, remains in the temporal table for validation.
2. particular referenced object is not valid and will not be used for update in the future at all, however, it is necessary to store historical data continuously along. In that case, archivation process is started to transfer data to the specific repository (*archive*). These activities are provided by the *Temporal Archive Master Process (TAM)* with its executors (*TAEn*) [17].

Principles are shown the fig. 10.

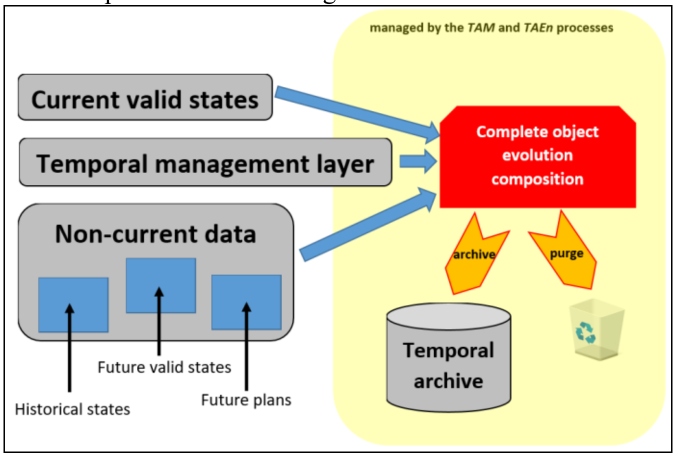


Fig. 10. Temporal Archive Master Process activities

The difference is based on using *purge* keyword. By default, archive process is used (fig. 11 – syntax).

```
Delete from table_name [{ archive | purge }];
```

Fig. 11. Removing object data syntax

From the logical perspective, script determines the temporal characteristics and processing environment. Before the *DDL* command execution, temporal declaration is identified and particular attributes or the whole tables are registered or de-registered using preprocessing layer, which identifies temporal extension of the commands and invokes *Temporal Registration Master Process (TRegMaster),* which is delimited by *Temporal Manager Process*, which is in charge of the entire temporal layer – fig. 12.

Fig. 12 shows also architectural model of the proposed solution – it is located in the temporal layer and interconnected to *Temporal Manager* process (yellow color box in fig. 12). As you can see, *Temporal manager* process invokes particular *Temporal Registration Master Process (TRegMaster)*, which provides required registration. It is done by the defined package (*DBMS_TEMPORALITY*) with the following methods:

- ***Register*** (add new attribute/table)
- ***Change*** (reflects existing definition)
- ***Remove*** (purge/archive)

In physical way, *Register* method of the *DBMS_REGISTRATION* package is associated with altering table, which can be influenced by the following options:

- *Alter table => ADD*
- *Alter table => DROP*
- *Alter table => MODIFY*

The first two command definitions can be processed directly, however, if attribute definition is to be changed, it cannot be reflected as direct change of attribute data type in temporal layer, because existing data type would be lost. Therefore, physical denotation of the *MODIFY* option is replaced by the *DROP* of the exiting attribute (which is consequently moved into the archive repository) and *ADD* – adding new attribute definition.

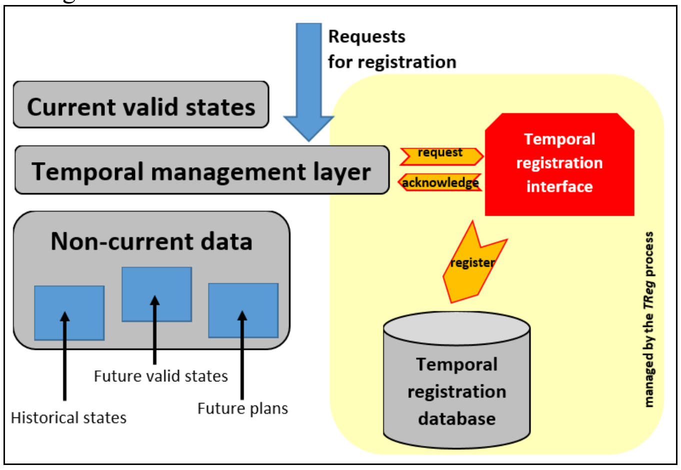


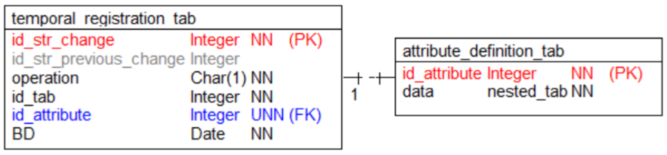
Fig. 12. Temporal registration

V. Physical Representation

For the physical representation, particular *Temporal Registration Master Process (TRegMaster)* must be launched. During the temporal database definition using our approach, it is started automatically immediatelly after *Temporal Manager* Pr*ocess* and becomes *inactive*. Status is changed to *active* only if the temporality definition is changed. Its aim is to to detect temporality from the *DDL* statements by the command decomposition. Afterwards, individual characteristics are registered. For these purposes, following table has been defined. As you can see, the structure is similar to attribute oriented architecture, the association is delimited by one attribute modelling interval – *BD. ED* can be calculated dynamically based on consecutive change. *Allen relationships* ensuring consistency are used for bordering previous definition [9] [10]. In principle, only relationship [*meets*] is used, whereas there can be no spaces (undefined states) inbetween. Each performed change of the model is triggered and enforced by current date delimiting previous valid structure definition. History and evolution can be obtained using linked list – *id_str_change* and *id_str_previous_change*. In theoretical way, also future structure redefinition operation can be used. In that case, structure would be automatically changed at the defined timepoint. However, it should be ensured, that no data states would be changed during the redefinition, otherwise such data would be lost.

Fig. 13 shows schema of the table used for storing structure and model over the time. Registration table itself consists of these attributes:

- *ID_str_change.*
- *ID_str_previous_change* – references the last change of the table/attribute identified by *ID_str_change*. It can also hold *NULL* value, which represents the process of adding (registering) new temporal table to the system.
- *Operation* – reflects the following options: *A* (add), *M* (modify), *D* (drop).
- *ID_tab* – references the table (each table has unique name (code) defined by the system and stored in data dictionary automatically).
- *ID_attribute* – carries information about added attribute – it is foreign key to the table, which holds the full attribute definition. The relationship type cardinality is 1:1.
- *BD* – begin data of the new state validity of the table specification.



temporal_registration_tab			
id_str_change	Integer	NN	(PK)
id_str_previous_change	Integer		
operation	Char(1)	NN	
id_tab	Integer	NN	
id_attribute	Integer	UNN	(FK)
BD	Date	NN	

attribute_definition_tab			
id_attribute	Integer	NN	(PK)
data	nested_tab	NN	

Fig. 13. Temporal registration data model

The fact that it was just a change of the type of attribute is expressed by a linked list of the changes (*id_str_previous_change*) in two layers. Main part (grey color in the fig. 14) represents the structural changes – adding or removing attribtues over the time. Changes on existing attributes are modelled for each attribute in separate deposit (blue color in fig. 14) – *NULL/NOT NULL, UNIQUE / DISTINCT*, *data types* characterizing *column* and *domain* integrity. Pointers are defined by the attribute *id_str_change* and *id_str_previous_change*.

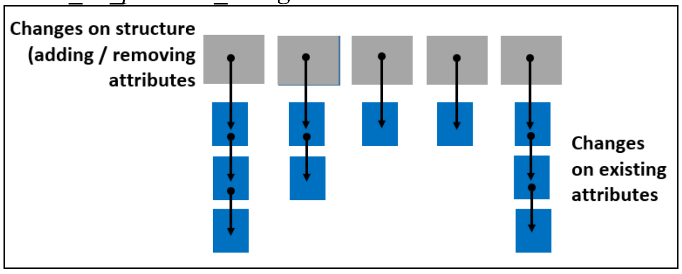


Fig. 14. Management of temporal structure

VI. Performance

Experiment results were provided using Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production; PL/SQL Release 11.2.0.1.0 – Production. Parameters of used computer are:

- Processor: Intel Xeon E5620; 2,4GHz (8 cores),
- Operation memory: 16GB,
- HDD: 500GB.

Experiment characteristics are based on real environment consisting of *1000* sensors producing data *ten times for one second*. If the difference between consecutive values is lower than *1%*, such values are not stored in the database and original value is considered as unchanged. Thus, based on our environment, average amount of new values is approximately *1000* per second. Amount of data after one hour is *3 600 000*.

The first part deals with the temporal structure definition using several approaches. First model (1) uses *object level temporal architecture* using *uni-temporality* defined by validity interval. Second model (2) uses pure solution using *Flashback backup and log files*, which is naturally the worst one, because of the necessity of data loading from the backup and activating stored log files. Notice therefore, that each backup as well as log file consists of the further set than data to be processed and retrieved, however, such data cannot be separated without full loading from binary files. Third model (3) is based on *Flashback repository* and *Flashback logs*. It provides robust solution, if image of the database should be provided at defined timepoint. However, it has significant weak side, if we need to get insights of the object changes over the time. Last – forth model is based on our *proposed temporal architecture*. In our solution, each temporal attribute data type is changed after 1 minute (totally performed 60 times).

Tab. 1 shows the provided results. Processing costs, CPU (%) and processing time (seconds) were measured using *autotrace* and *timing* functionality of the DBS Oracle.

For the evaluation, *object level temporal architecture* will be used as reference model – 100%. *Approach* (2) based on *backup* and *logs* is significantly worse, because of the loading necessity – slowdown more than 145% for costs, 155% for CPU and 148% for processing time. Therefore, such solution is totally inappropriate. On the other side, significant improvement provides *Flashback technology* – model (3) – improvement of 58% for costs, 59% for CPU and 59% for processing time. If the processing would deal with only database images and snapshots, performance would be much better – approximately 1% slowdown in comparison with model (4) performance (attribute oriented approach with registration functionality).

The best solution provides *attribute oriented model* (model 4), which can process and manage data model changes over the time. Global performance of the model (4) is following (reference is model (1)):

• Costs	80%
• CPU	76%
• Processing time	87%

TABLE I. PROCESSING RESULTS

	Object level (1)	Backup & log files (2)	Flashback repository (3)	Attribute oriented approach + registration (4)
Costs	17 011	41 824	7 085	3 252
CPU (%)	52	133	21	12
Processing time (s)	257,5	640,8	103,3	32,8

Fig. 15 shows the graphical representation of the results – processing time.

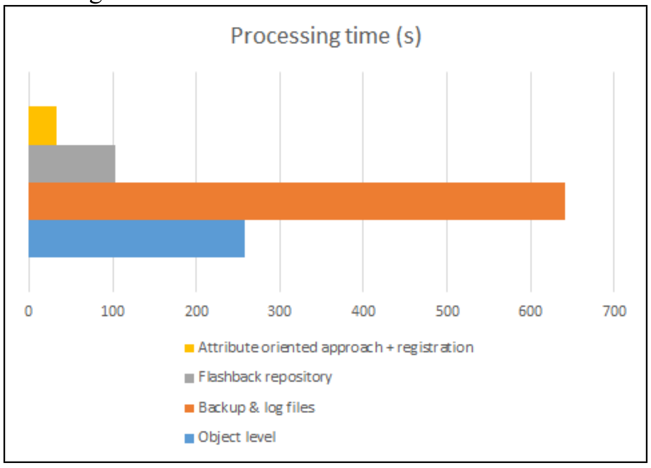


Fig. 15. Processing time (s)

Process of the registration based on our environment lasted totally 0,075s to the whole structure (in this case, only one master process has been used with no executors).

The second part of the experiments deals with the impact of the processing based on number of executor processes - *Temporal Registration Executor Process* (*TRegExecn*). Adding executor process significantly influence performance by reducing processing costs. However, adding too many executor processes does not provide sufficient power because of the synchronization necessity and resource competition. Based on experiments, limitation is 10 processes (tab. 2, fig. 16).

TABLE II. PROCESSING COSTS BASED ON NUMBER OF EXECUTORS

	Number of executor processes					
	1	2	5	10	50	100
Processing costs (%)	100	68	26	14	13	13

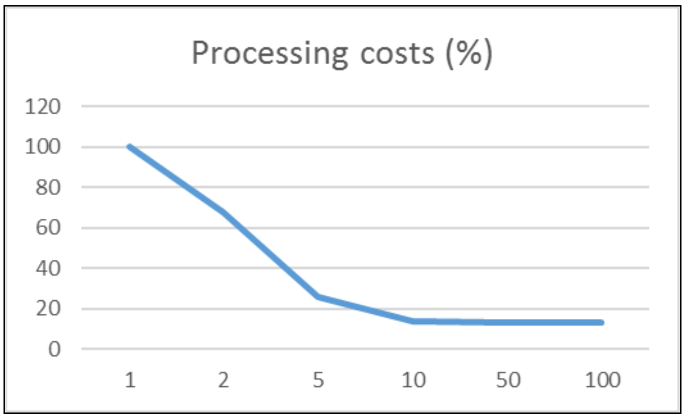


Fig. 16. Processing costs based on number of executors

VII. Conclusions

Each data tuple is delimited by the time validity from the definition. It´s on developers and users, whether such definition will be stored or not. Temporal database concept proposes management technology for data evaluations and storage possibilities. Performance of the data retrieval retrieval is a crucial part highlighting efficiency. Several solutions have been proposed for dealing with time bordered states depending on the granularity and frequency of changes. Conpect of using backups and log files for the temporal data evaluation has been significantly improved by the *Flashback repository,* which allows really efficient solution for deadling with images and snapshots of the database at defined timepoint, however does not provide sufficient power for monitoring changes over the time. Our proposed temporal solution core part is based on attribute granularity, which can be optionally formed into groups, if some data portions are synchronized.

Storing data over the long time period must deal and react to the possibilities of changing physical structure of the database – development of the data approach and model as well. It covers an option to add or remove the whole table, change the definition of the attribute or the group of attributes to become temporal. For these purposes, new processes have been introduced to reflect and manage such changes and reflect them into temporal access layer. Thanks to that, images over the time with regards on the structure at given time can be obtained. As we have shown, proposed solution is complex, secured by the new layer, which manages structure in the defined temporal table structure.

In the future, we will focus on the development of temporal registration management in the distributed environment with emphasis on the workload rebalancing on individual nodes in the temporal sphere.

Acknowledgment

This publication is the result of the project implementation:

Centre of excellence for systems and services of intelligent transport, ITMS 26220120028 supported by the Research & Development Operational Programme funded by the ERDF and *Centre of excellence for systems and services of intelligent transport II.,* ITMS 26220120050 supported by the Research & Development Operational Programme funded by the ERDF.

This paper is also supported by the following project: "*Creating a new diagnostic algorithm for selected cancers*," ITMS project code: 26220220022 co-financed by the EU and the European Regional Development Fund.

"Podporujeme výskumné aktivity na Slovensku
Projekt je spolufinancovaný zo zdrojov EÚ

References

[1] Q. Abbas, H. Shariq, I. Ahmad, S. Tharanidharan, “Concurrency control in distributed database system”, 2016 International Conference on Computer Communication and Informatics (ICCCI)

[2] K. Ahsan, P. Vijay. “Temporal Databases: Information Systems”, Booktango, 2014.

[3] L. Ashdown. T. Kyte “Oracle database concepts”, Oracle Press, 2015.

[4] G. Avilés et all. “Spatio-temporal modeling of financial maps from a joint multidimensional scaling-geostatistical perspective”, 2016. In Expert Systems with Applications. Vol. 60, pp. 280-293.

[5] R. Behling et all., “Derivation of long-term spatiotemporal lanslide activity – a multisensor time species approach”, 2016. In Remote Sensing of Environment, Vol. 136, pp. 88-104.

[6] C. J. Date, N. Lorentzos, H. Darwen. “Time and Relational Theory : Temporal Databases in the Relational Model and SQL”, Morgan Kaufmann, 2015.

[7] R. Heckman, “Data layer”, Wiley-IEEE Press, 2016.

[8] J. Chomicki, J. Wihsen, “Consistent Query Answering for Atemporal Constraints over Temporal Databases”, 2016

[9] T. Johnston. “Bi-temporal data – Theory and Practice”, Morgan Kaufmann, 2014.

[10] T. Johnston and R. Weis, “Managing Time in Relational Databases”, Morgan Kaufmann, 2010.

[11] A. Kadir and N. Adnan, “Temporal geospatial analysis of secondary school students´ examination performance”, 2016. In IOP Conference Series: Earth and Environmental Science, Vol 37, No. 1.

[12] D. Kuhn, S. Alapati, B. Padfield, “Expert Oracle Indexing Access Paths”, Apress, 2016

[13] M. Kvassay, E. Zaitseva, J. Kostolny, and V. Levashenko, “Importance analysis of multi-state systems based on integrated direct partial logic derivatives”, In 2015 International Conference on Information and Digital Technologies, 2015, pp. 183–195.

[14] M. Kvet, K. Matiasko, “Improving performance of database system using architectural layer”, CISTI 2016.

[15] M. Kvet, K. Matiasko, M. Kvet, Temporal index dispatcher layer service for intelligent transport system”, Elektro 2016.

[16] M. Kvet, M. Vajsova, „Performance study of the index structures in audited environment“, ICITST-2016, 2016.

[17] M. Kvet and K. Matiaško, “Temporal Data Group Management”, *unpublished*.

[18] D. Kuhn, S. Alapati, B. Padfield, “Expert Oracle Indexing Access Paths”, Apress, 2016.

[19] R. Niemiec, “Oracle Query Tuning”, Oracle Press, 2014.