# Temporal Extension of the Select Statement – New Clauses

Michal Kvet
University of Zilina
Zilina, Slovakia
Michal.Kvet@fri.uniza.sk

Karol Matiaško
University of Zilina
Zilina, Slovakia
Karol.Matiasko@fri.uniza.sk

***Abstract*—Effective timed data processing belongs to one of the most important tasks in the development of current information and database systems. It is not, however, only the changes in time management, but also the complex record of changes during the whole life cycle of the object – historical values, actual states, but also data valid in the future. Existing temporal solutions are inadequate in terms of performance - effectiveness of the whole system, which is manifested by the size of the required data and processing time. There is no temporal solution for Select statement defined and the user has to manage it explicitly. This paper deals with the principles of temporal data modeling on the object and attribute level. It also describes the characteristics of Select statement used in the temporal system, extends it with new characteristics and defines new layer for transformation into existing syntax.**

## I. Introduction

Massive development of data processing requires access to extensive data using procedures and functions to provide easy and fast manipulation. The basis is the database technology [11].

Database systems are the root of any information system and are the most important parts of the information technology. They can be found in standard applications, but also in critical applications such as information systems for energetics, industry, transport or medicine. The development of data processing has brought the need for modeling and accessing large structures based on simplicity, reliability, and speed of the system. However, even today, when database technology is widespread, most databases process and represent current valid data. However, conventional relational database definition can be extended for temporal data modeling, which allows changes, evolution monitoring, process optimization, prognoses and analyses creation [1], [12], [13], [14].

Complex data management can be found almost in any field, many systems are based on temporal logic [5], [17], [21] and extension management (paradigm) [18]. Temporal data processing must deal with data quality [20], security [19] and reliability aspect.

This paper deals with the temporal architecture and defines problems of data selection, therefore new layer for temporal data manipulation has been developed and the *Select* statement has been extended. Extended parts are characterized by the emphasis of the current relational database transformation. It is divided into nine sections. Technical background and theory are described in section 2, followed by new clauses options (section 3), structurally and implementation-oriented definition covered by the sections 4 up to 7 (section 4 - TYPE_OF_GRANULARITY, section 5 – EVENT_DEFINITION, section 6 – EPSILON_DEFINITION, section 7 – MONITORED_COLUMN_LIST). Section 8 deals with the computational study. The last section concludes the developed solution and proposes expansion options.

## II. Theory

The temporal system has been developed soon after the development of databases. In the first phase, historical data were saved using log files and archives. Thus, historical data could be obtained, but it is a complicated process, these data are in the raw form and handling them was difficult, required too much time. The main problem was data operation loss if the backup frequency was not suitable. Thus, decisions based on historical data could not be used, because large backup images had to be loaded manually, which took a long time. Another problem is the impossibility of future valid data modeling and management [1], [6], [7], [8].

Later, the temporal systems have been developed defining a new paradigm for selecting one or more rows based on the specified criteria, for projecting one or more columns to the output sets and for joining the tables by specifying relationship criteria. It means that individual operations must contain also time definition. This paradigm is still valid.

The first model (conventional approach) in Fig. 1 does not use the time for definition at all, it cannot provide management for non-current data in the main structure. The primary key is defined by the attribute *ID* (can be composite). In the non-timed table, each row represents a specific instance identified by a primary key. The uniqueness of the primary key values without defining additional conditions ensures that the number of rows in the table is identical with the number of managed objects. Any change has directly reflected the database and the old value is deleted.

The second model is a uni-temporal system, *ID* is a unique identifier; *PK* refers to a primary key. *BD* and *ED* is a pair of columns defining the beginning and end value of the period – validity. The uni-temporal system always uses the composite primary key – object identifier and time interval defining the validity state characterizing the row [2], [3], [4], [7], [8].

A special type of uni-temporal system consists of the only a one-time attribute (*begin date – BD*) that is part of the primary key (third model in Fig. 1). This means that any change of the corresponding object determines the validity of the prior state. The principles of transformation uni-temporal system defined

ISSN 2305-7254

by the begin date of the validity to the standard approach and time intervals are described in [8], [17]. The special approach has been developed by us, which is based on the attribute granularity, not the entire object. Thus, if the value of the single attribute is changed, only that value is updated (not the whole state, which consists of a various number of temporal columns) [6], [9], [15].

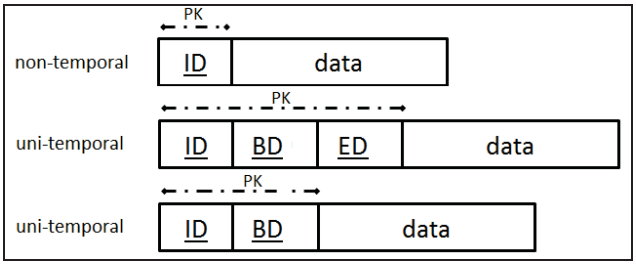


Fig. 1. Conventional and object level temporal model

## III. TEMPORAL SELECT STATEMENT

The *Select* statement in the relational database approach is considered as the most important and most frequently used SQL statement based on performance. With this statement, the desired data can be obtained from the database by using relational tables. The basic syntax of the *Select* statement in the conventional database consists of these six parts - *Select*, *From*, *Where*, *Group by*, *Having* and *Order by*.

Although conditions can be defined in the *Where* clause, this segment does not cover the complexity and structure of the temporal system. Therefore, in the following section, we describe our proposed new ways to enhance the whole concept of the management of temporal data.

We designed and implemented syntax, which shows the temporal extension of the *Select* statement using these parts:

- *EVENT_DEFINITION,*
- *EPSILON_DEFINITION,*
- *MONITORED_COLUMN_LIST,*
- *TYPE_OF_GRANULARITY.*

Our proposed extended clauses and the whole Select statement opportunities are presented in the following code. The importance of individual clauses is subsequently described in the next sections.

```
[CREATE TABLE table_name AS]
  SELECT [ALL | DISTINCT | UNIQUE]
    { * | attribute_name | function_name [(parameters)]}
        [,...]
   FROM table_name [alias] [,...]
    [WHERE condition]
         [EVENT_DEFINITION]
         [EPSILON_DEFINITION]
         [MONITORED_COLUMN_LIST]
     [GROUP BY attribute_list]
      [HAVING condition]
         [TYPE_OF_GRANULARITY]
       [ORDER BY column_name [ASC | DESC] [,...]]
```

## IV. TYPE_OF_GRANULARITY

Our first introduced clause is *a TYPE_OF_GRANULARITY* expression, which is included after the *Having* clause. It is a way of formatting the final set, defines the sensitivity and details of the changes. There can be three types defined. *Object* - defines the granularity on object level – list of object states changes regardless of the type of change. Changed attribute values themselves can be taken using *Column* characteristics. The difference between the options shown in the following figure. At time $t_1$, object $O_1$ state is changed from $S_1$ state to $S_2$ state. If the *Object* definition is used, we get information about the change of the object $O_1$. However, specific new values (new value of the attribute $H = H_{A12}$) are obtained through the clause *Column.*

The access rule *COLUMN_CHANGES_MONITORING* displays also comprehensive information – new ($H_{A12}$) and also the previous value of the attribute ($H_{A11}$) – Fig. 2.

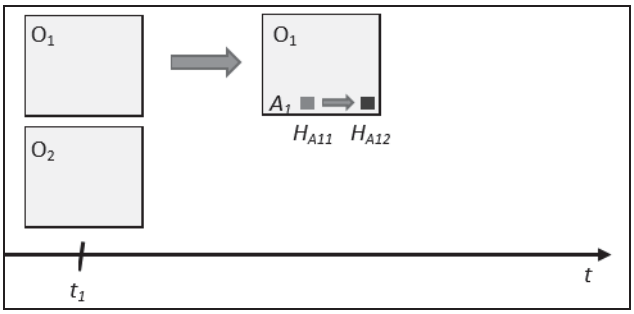


Fig. 2. Update of the state of the object O1

Next code shows the output of the *Select* statements with respect to the granularity access rules.

```
OBJECT: t1 – state of the object O1 has been changed.
COLUMN: t1 – state of the object O1 has been changed.
                    Attribute A1 now has value HA12.
COLUMN_CHANGES_MONITORING: t1 - Attribute A1 of the object O1
                           has been changed.
                    Old value = HA11 ; new value = HA12.
```

Following code describes the principle of transformation to the existing syntax.

```
OBJECT:
  -- the only identifier of the object (object_id) and time
point of the change (ch_timepoint) is selected.
        select object_id, ch_timepoint ...

COLUMN:
  -- identifier of the object (object_id), timepoind of the
change (ch_timepoint) and new value (new_val) is selected.

        select object_id, ch_timepoint, new_val
         from ...
          where new_val != old_val ...

COLUMN_CHANGES_MONITORING:
  -- identifier of the object (object_id), timepoind of the
change (ch_timepoint), old (old_val) and new value (new_val)
is selected.

        select object_id, ch_timepoint, new_val, old_val
         from ...
          where new_val !=old_val ...
```

Default selection criterion is *Column*. If this clause was not used, we would get all temporal attribute values regardless the change of them at defined timepoint (interval).

An essential part of the state monitoring is to get direct predecessor (state) for changes monitoring. Following code shows the algorithm for obtaining previous change if exists.

The solution is based on using *With* clause, which eliminates multiple same table definition.

```
With tab_with (
        Select t2.*
         from TAB1 t1. TAB2 t2
          where t1.ID = p_state.ID
             AND t1.BD = p_state.BD
              AND t2.ED <= t1.BD)
Select *
    from tab with
           where ED = (select max(ED)
                                from tab);
```

## V. EVENT_DEFINITION

Our proposed syntax definition covers also *EVENT_DEFINITION* part, which extends the *Where* clause of a *Select* statement. It specifies a range of time and processed data obtained by way of a point in time (*defined_timepoint*) - just a reference to the point in time - or time interval (*defined_interval*), the definition of which includes two-time values - the begin and end time point of the validity.

In the database instance session, the user can set the type of used interval (closed-closed, closed-open representation). However, it can be redefined directly for the executed *Select* statement. Thus, the second (optional) parameter is the interval type to be used (*CC* - closed-closed, *CO* - closed-open type).

```
defined_timepoind(t)
defined_interval(t1, t2, [CC | CO])
```

Fig. 3 shows the input interval with the closed-closed characteristics, Fig. 4 shows the closed-open representation defined inside the method. Input time interval (*defined_interval* ($t_1$, $t_2$)) is expressed in the top part of the figure, then the states including output set based on real representation in the database are highlighted.

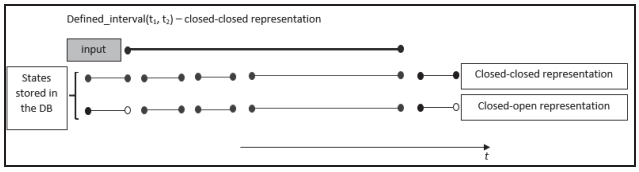


Fig. 3. Closed-closed input representation

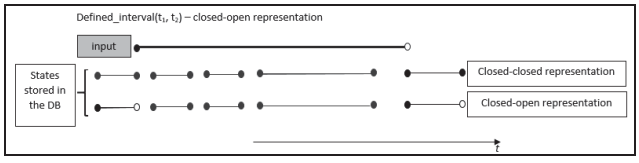


Fig. 4. Closed-open input representation

The following block shows the transformation to an existing *Select* statement syntax. We assume that the object state is bordered by the beginning of the period (*BD*) and end date (*ED*). Terms $t_1$ and $t_2$ represent the boundary of the monitored interval (*defined_interval*):

```
defined_timepoint(t):
--for closed-closed time validity representation <BD, ED>.
                where BD <= t AND t <= ED ...

--for closed-open time validity representation <BD, ED>.
                where BD <= t AND t < ED ...

defined_interval (t1, t2, CC | CO):
   closed-closed input interval representation <t1, t2>:
     Validity modelled using closed-closed representation:
                           where BD <= t2 AND ED >= t1 ...
     Validity modelled using closed-open representation:
                           where BD <= t2 AND ED > t1
   closed-open input interval validity <t1, t2>:
     Validity modelled using closed-closed representation:
                           where BD < t2 AND ED >= t1 ...
     Validity modeled using closed-open representation:
                           where BD < t2 AND ED > t1
```

## VI. EPSILON_DEFINITION

For the purposes of changes and progress monitoring over the time, it is convenient to define rules that affect the size of the output processed sets. We developed *EPSILON_DEFINITION* as the determination of the method by which it is possible to filter out irrelevant changes, especially in sensor data. Each referenced temporal attribute may have defined the precision – relevance – the minimal value of the significant change - Epsilon ($\varepsilon$) value. If the difference between two consecutive values of the attribute is less than the value of the Epsilon ($\varepsilon$) parameter defined for the corresponding temporal column, this change will not appear in the result set returned by the *Select* statement. If this clause is not used, then the default value of the minimum change *($\varepsilon$ = 0)* is used. Thus, any change will be processed regardless of the relevance.

To use this functionality, it is necessary to define a function that will map values of the non-numerical data types to the types in order to quantify the change. For each data type, we need to define a *Map* function (or use implicit conversion function). The input parameter of this function is the value of the primary data type, the result returned is a numeric value (integer, float, long, ....):

```
Create or replace map function f_date_type(val data_type)
return longint
is
begin
  return transformed_value (val); -- into longint;
end;
/
```

In Fig. 5, we characterize the use of the *Epsilon* ($\varepsilon$) principle for the definition expressed by the medically processed data – brain tumor detection. The input values are first filtered based on the position (only areas, where anomaly, respectively tumor can be located, are monitored). Then the marker values are compared during the time evolution (tumor markers means a substance, usually a protein, the occurrence of which indicates the presence of cancer in the patient´s body [16]). If the new marker value ($_{new}$) expresses significant change:

$$\left| m_{new} - m_{old} \right| >= \varepsilon$$

specific algorithms check incorrect positivity. If not detected, the value is stored in the database. This process reduces the amount of stored data.

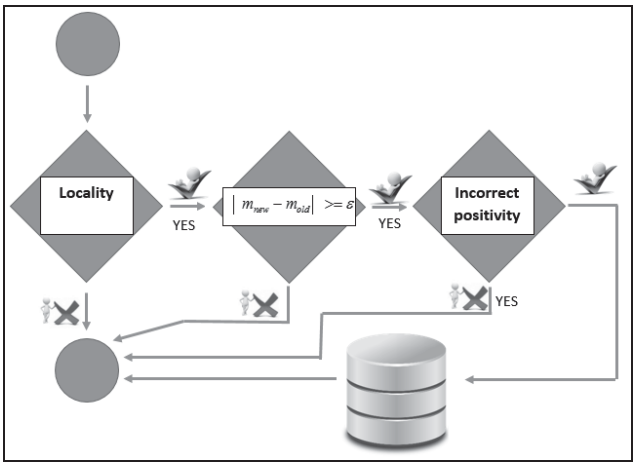


Fig. 5. Incorrect positivity (brain tumor detection)

Fig. 6 shows the marker values without using Epsilon ($\varepsilon$) definition, whereas Fig. 7 uses this approach - ($\varepsilon = 0.5\%$). Note that if you do not use Epsilon ($\varepsilon$) definition, a complete image is stored. However, by using Epsilon ($\varepsilon$) principle, not processed values are replaced by evaluated areas from the past. The thick lines border region of interest (which should be monitored over the time).

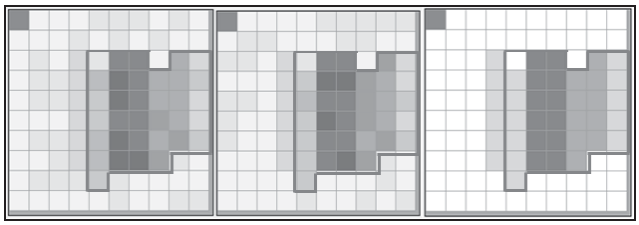

Fig. 6. Solution without using Epsilon (ε) approach

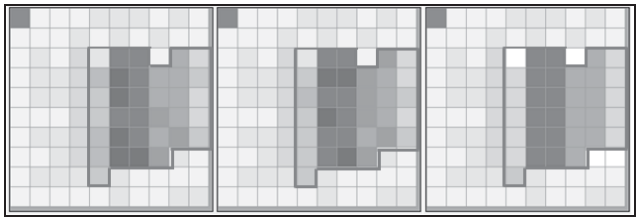

Fig. 7. Solution using Epsilon (ε) principle

The following block illustrates the transformation to the existing *Select* statement syntax:

```
--implicit conversion:
where ABS(new_val - old_val) >= Epsilon

--explicit conversion:
where ABS(f_data-type(new_val) - f_data-type(old_val)) >= Epsilon
```

In this approach, however, specific situation can occur – attribute value change rate is high, but the difference between actual and previous value is lower than the *Epsilon* ($\varepsilon$) parameter. Thus, in the global view, the progress reflected by the time $t_1$ can be significant. The problem also expresses the following figure.

Suppose attribute value $A = 5$ (represented at time $t_1$) and Epsilon temporal change parameter value $\varepsilon = 1$. Fig. 16 shows the progress of changes - observed value is still growing, but not so much, thus the neighboring changes difference is not greater than defined parameter ε (Fig.6).

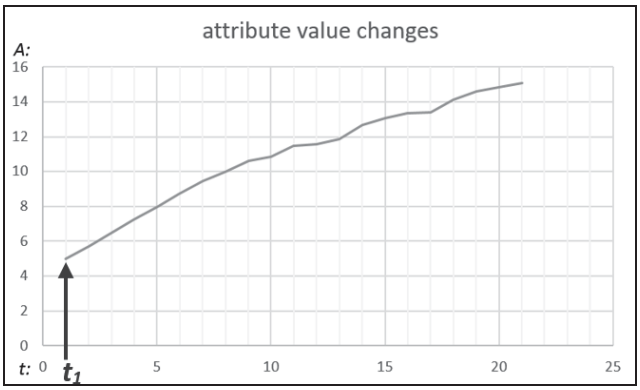


Fig. 6. Time changes evolution

Therefore, *EPSILON_DEFINITION* clause allows you to define not only management for neighboring changes. Another optional parameter is the size of a controlled time frame (*time_frame*). Each column in the temporal clause *EPSILON_DEFINITION* is characterized by the parameter Epsilon ($\varepsilon$) and also time frame. When defining the time frame, you need to specify the method of processing - the smallest granule – time point or attribute change:

```
EPSILON_DEFINITION_attribute A1 (ε1, [time_frame1],
                                 [frame_type1],
                                 [referential_value1]),
EPSILON_DEFINITION_attribute A2 (ε2, [time_frame2],
                                  [frame_type2],
                                  [referential_value2]),
...
```

The last option of this method is the fixed (global) referential value, thus processed frame is still growing during the time.

## VII. MONITORED_COLUMN_LIST

Our proposed clause *MONITORED_COLUMN_LIST* as the extension of the *Select* for temporal approach allows you to define a list of columns definition, which is relevant for processing and should be monitored. This list does not need to be identical to the first part of the *Select* statement, however, it can consist only of the temporal attributes (not conventional or functions):

```
MONITORED_COLUMN_LIST(attribute1, attribute2,
                      attribute3, ...)
```

If there is a change of at least one column defined in clause *MONITORED_COLUMN_LIST*, resulting command will reflect this change. If the clause is not specified, it will automatically be replaced by monitoring all temporal attributes:

```
MONITORED_COLUMN_LIST(*) -all temporal columns monitoring
```

## VIII. EXPERIMENTS

Our experiments and evaluations were performed using a medical information system – measured and processed parameters were performed using volunteers – long-term magnetic resonance imaging (MRI) results monitoring [10], [16].

All experiments were provided using the Oracle 11g database system. In this part, a total number of records in the main structure was 10 000, each record contains 10 MRI results.

In this computational study and performance limitation section, we highlight time to get required data and size of the structure. We compare our implemented temporal solution on column level with the standard uni-temporal system followed by the extension of the transaction management of the developed system compared to bi-temporal structure (uni-temporal approach on object level extended by the definition of the transaction time validity).

Although in general, we are talking about transaction management, processing, in this meaning, it is based on measurement error reduction. If it is possible to reduce measurement error, new transaction inserts the corrected marker value into the database obtained by the approximation and monitoring the progress and dependencies of marker values over time.

The first model deals with the standard uni-temporal approach (reference 100%) based on object level. In comparison with the temporal structure on column level (model 2 and 3), there is a significant acceleration of the system:

***without*** *using Epsilon (Ɛ) principle (model 2):*

- Size: *41,60%.*
- Time to get the current image ($T_1$): *73,88%.*
- Time to get all life-cycle MRI data result: *59,98%.*

*using Epsilon (Ɛ) principle (model 3):*

- Size: *51,55%.*
- Time to get the current image ($T_1$): *73,88%.*
- Time to get all life-cycle MRI data result: *65,97%.*

The last ($4^{th}$) model represents transaction processing modeled by the extension of the primary key using transaction time definition, it provides the slowest performance.

Fig. 8 shows the implemented results dealing with the size of the whole database structure, Fig. 9 deals with the processing time monitoring the current data image and time to get the whole evolution of the object, as well.

Epsilon ($\mathcal{E}$) parameter has been used, which separates the objects of interest (anomalies, tumors) from other brain tissues. However, it is very important to set the appropriate value of the parameter. Too high value can cause the reduction of the potential anomalies from the output image [16]. Fig. 10 shows the size – standard approach and epsilon value dependency. If the system for incorrect positivity detection is used, the size requirement is lower.

Experiment results can be shown on the Table I. It is expressed by the obtained values and monitored parameters (time and size). First of all, when comparing granularity, pure column level temporal architecture (model 2) reaches the improvement of 41,61% for size, 73,88% for getting actual data image and 59,98% for accessing data evolution (reference – model 1 – 100%). Epsilon approach can reduce the data amount, which is not relevant. Based on our experiment scene, model 3 got the 17,03% improvement for size management (reference – model 2) and 14,96% for obtaining object data changes. The worst performance score made a bi-temporal solution, which manages also data changes. In comparison with validity aspect of model 1, the slowdown is 26,96% for the process of getting current data image and 21,67% for monitoring object evolution over the time. When dealing with structure size, it requires an increase of 11,89% (reference – model 1 – 100%).

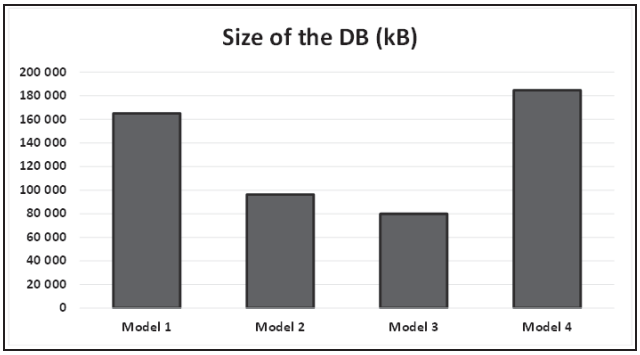


Fig. 8. Size of the DB

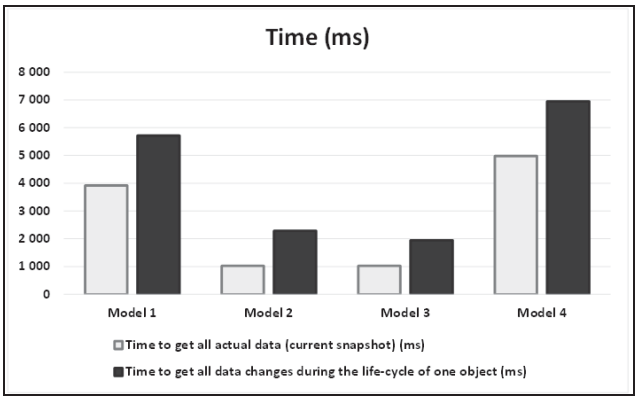


Fig. 9. Results - time

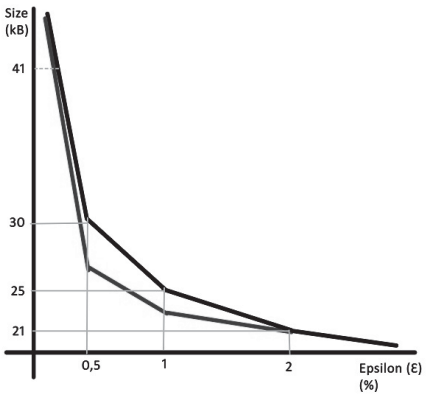


Fig. 10. Epsilon value dependency

TABLE I. EXPERIMENT RESULTS

| | Uni-temporal system | Uni-temporal system | Uni-temporal system | Bi-temporal system |
|---|---|---|---|---|
| | Object level | Column level | Column level | Object level |
| | --- | --- | Epsilon approach | --- |
| | Model 1 | Model 2 | Model 3 | Model 4 |
| Size of the DB (kB) | 164 940 | 96 312 | 79 912 | 184 552 |
| Time to get all actual data (current snapshot) (ms) | 3 921 | 1 024 | 1 024 | 4 978 |
| Time to get all data changes during the life-cycle of one object (ms) | 5 712 | 2 286 | 1 944 | 6 950 |

## IX. CONCLUSION

A conventional database object is represented by one row – current state of the object, whereas the temporal database system offers processing object valid data and their changes and progress in time. Data processing in the temporal environment requires access to the whole information about the evolution of the states during the life-cycle. Effective managing temporal data is the core of the development and can be used for decision making, analyses, process optimization, which is a very significant factor in the industrial environment.

The problem of the current temporal paradigm is weak support for data management. Simply, temporal systems proposed in the recent past do not offer sufficient power to manage large volumes of data with emphasis to reliability and effectiveness of the statements. Most significant is the *Select* statement, which can be considered as the main part of the *DML* statements, also the critical performance factor of the whole system. The new definition of the *Select* statement extends the conventional principle by adding clauses for time management. In sensor data processing, the problem is more visible due to the precision of the measured data. Therefore, layer with Epsilon ($\mathcal{E}$) principle has been defined. Moreover, all new clauses can be directly transformed into existing syntax.

Temporal data processed over the time are usually large. The processing requires sophisticated access methods. In the future, we will focus on various index structure creation, index distribution, which can improve the performance of the system, too. *Select* statements to be executed will be transformed based on index structures to improve the performance of the system.

## 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.

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] C. J. Date, *Date on Database*. Apress, 2006.

[2] C. J. Date, "Logic and Databases – The Roots of Relational Theory", Trafford Publishing, 2007.

[3] C. J. Date, H. Darwen, and N. A. Lorentzos, *Temporal data and the relational model*, Morgan Kaufmann, 2003.

[4] P.N. Hubler and N. Edelweiss, "Implementing a Temporal Database on Top of a Conventional Database", 2000. Conference SCCC '00, pp. 58 – 67

[5] N. Chumakova, V. Olenev and I. Lavrovskaya, "Conformance Testing of the STP-ISS Protocol Implementation by Means of Temporal Logic", 2017. Conference FRUCT 21.

[6] Ch. S. Jensen, "Introduction to Temporal Database Research",

[7] Web: http://infolab.usc.edu/csci599/Fall2001/paper/chapter1.pdf - Online January 2015.

[8] Ch. S. Jensen and R. T. Snodgrass, *Temporally Enhanced Database Design*, MIT Press, 2000.

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

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

[11] M. Kvet and K. Matiaško, "Transaction Management", 2014. CISTI, Barcelona, pp.868-873.

[12] M. Kvet and M. Vajsová, "Transaction Management in Fully Temporal System", 2014. UkSim, Pisa, pp. 147-152.

[13] M. Kvet, K. Matiaško, M. Kvet, "Complex time management in databases", In Central European Journal of Computer Science, Volume 4, Issue 4, 2014, pp. 269-284.

[14] P. Lewis, A. Bernstein, and M. Kifer, *Databases and Transaction Processing (An Application Oriented Approach)*, Addison-Wesley, 2002.

[15] J. Maté, "Transformation of Relational Databases to Transaction-Time Temporal Databases", in ECBS-EERC, 2011, pp. 27-34.

[16] M. T. Oszu and P.Valduriez, "Principles of Distributed Database Systems", Inria, 1991.

[17] O. Pianykh, "Digital Imaging and Communications in Medicine", Springer, 2008.

[18] R. Snodgrass, *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann Publishers", San Francisco, 2000.

[19] N. Teslya and I. Ryabchikov, "Blockchain-base Platform Architecture for Industrial IoT", 2017. Conference FRUCT 21

[20] X. Wei et all, "Data Quality Aware Task Allocation with Budget Constraint in Mobile Crowdsensing", 2018. In IEEE Accessm Volume 6

[21] H. Zhang et all, "End-to-end temporal attention extraction and human action recognition", 2018. In Machine Vision and Applications, Volume 29, Issue 7.