Managing Undefined Values in Temporal Environment

Michal Kvet, Karol Matiaško
Faculty of Management Science and Informatics, Department of Informatics
University of Zilina
Zilina, Slovakia
Michal.Kvet@fri.uniza.sk

ABSTRACT: *Database systems and temporal extension is core part of the data and information system processing. However, temporal paradigm has never been standardized. Object level temporal approach is based on extension of the primary key, whereas our approach uses column granularity, thanks to that, sensorial data with various granularity and frequency of changes can be effectively managed and accessed. Moreover, significant performance parameter is just index structure and mostly used B+tree. Such developed index cannot manage and deal with NULL values, therefore in this paper, we propose effective solution for storing undefined attribute values as well as time attributes delimiting validity.*

Keywords: Temporal Characteristics, Attribute Oriented Approach, Unlimited Validity, MaxValueTime, UndefTypeValue

Received: 12 June 2017, Revised 10 July 2017, Accepted 27 July 2017

I. Introduction

Information system management and development requires universal access to data, which should be evaluated, processed, stored and also retrieved. Such system must ensure robustness, effectivity and security. Soon after developing first database system, strong pressure on performance could be perceived. Architecture and approaches have been changed significantly over the years, however, the main paradigm based on conventional approach is still used. It is based on storing and manipulating only actual valid data. Thus, historical data are not stored in main structure at all, although they can be found in backup and log files. To provide complex security, most database systems work in archive log mode – all online redo log are archived. If they were not deleted automatically directly after creating new backup, it would be possible to reconstruct any object state during whatever time point or time interval. Quality, performance and reliability of such system would be really poor and questionable. Therefore, after development of first database systems, pressure for time data processing was strong resulting in developing new concepts, paradigms and management of such data. Unfortunately, it has not been certified and approved as standard, yet, so development stream was completely cancelled in 2001. Although later, there have been some temporal approaches [1] [2] [3] [9], most of them were based on object level temporal architecture. It is based on primary key extension. Object itself is then determined by its identifier as well as time delimiting particular object state. Uni-temporal system uses validity time extension, bi-temporal approach is based on validity, but consists also transaction validity frame. In general, we can use muti-temporal system, which can deal with various number of temporal spheres to be processed like validity, transaction validity, time position, location, reliability and many other factors. Fig. 1 shows the object level temporal model. It has, however, many limitations. First of all, object state consists of all processed attributes, thus, if some attribute value is not changed and some others are, duplicates must be stored. Moreover, there is strict limitation, if new attributes should be added to the object creating state. There is problem with data definition before the timepoint of adding new attributes. Another aspect is data model complexity. Whereas foreign key always references primary key or unique index, which forms also primary key, referential integrity management is too complicated with regards on state covering by time.

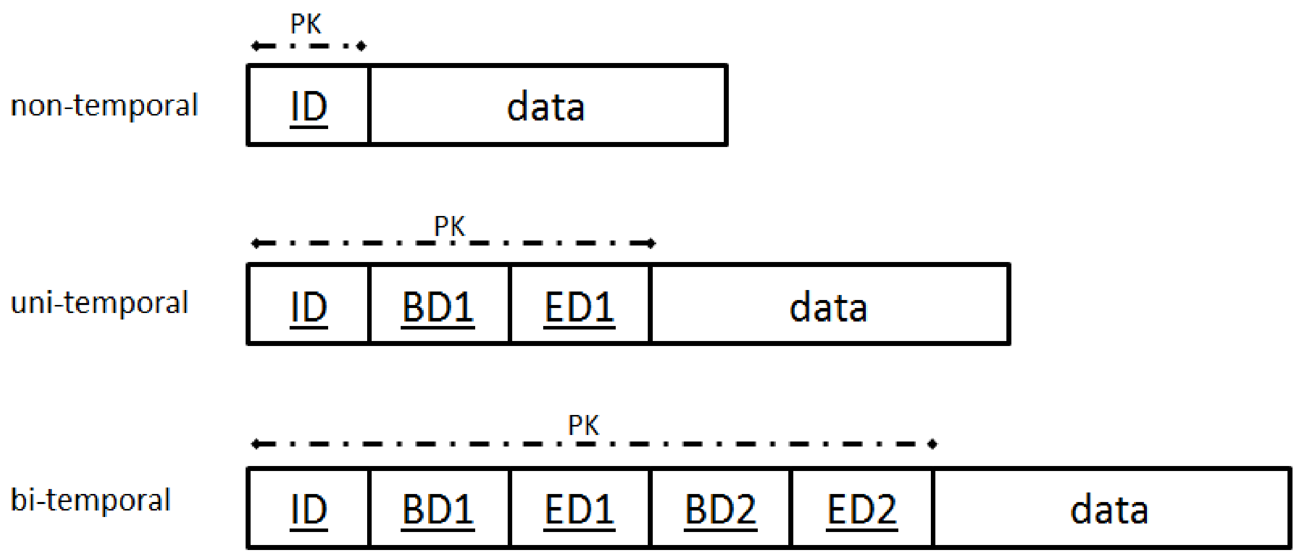


Figure 1. Temporal object model

First part of this paper deals with time spectrum supported by object level temporal architecture. Later, we concern on undefined values management covered by index structure to optimize data retrieval process and its performance. Problem of undefined value management is divided into two parts, which are handled differently – time (validity) and standard attribute. Then we propose column level temporal architecture, which has been developed for different frequency and granularity of attribute changes, sensorial data. It proposes complete structure and approach restructuralization and performance improvements. Temporal logic is covered in [5] [15] [21].

2. Time Spectrum Modelling Techniques

Time in the database can be modelled using one attribute (time point) or using time interval.

2.1 Time Point

Time point is characterized by only one attribute value determining time moment based on granularity. Such defined approach is, however, very inefficient, because each moment requires inserting new row defining state. Therefore, time point is this case expresses time limitation, mostly the first time of the validity of particular state. Thus, direct newer state is also border of previous state validity. Moreover, such architecture as core part of the temporal processing used today, does not support undefined values processing or even complete undefined or unknown states. Imagine *NULL* values, they do not determine undefined value in temporal sphere, they can have specific sense based on application domain. As we will describe later, *NULL* values defining validity interval can cause significant problem in terms of performance and complex evaluation. Another modelling method for the time spectrum interval is just time interval.

2.2 Time Inverval

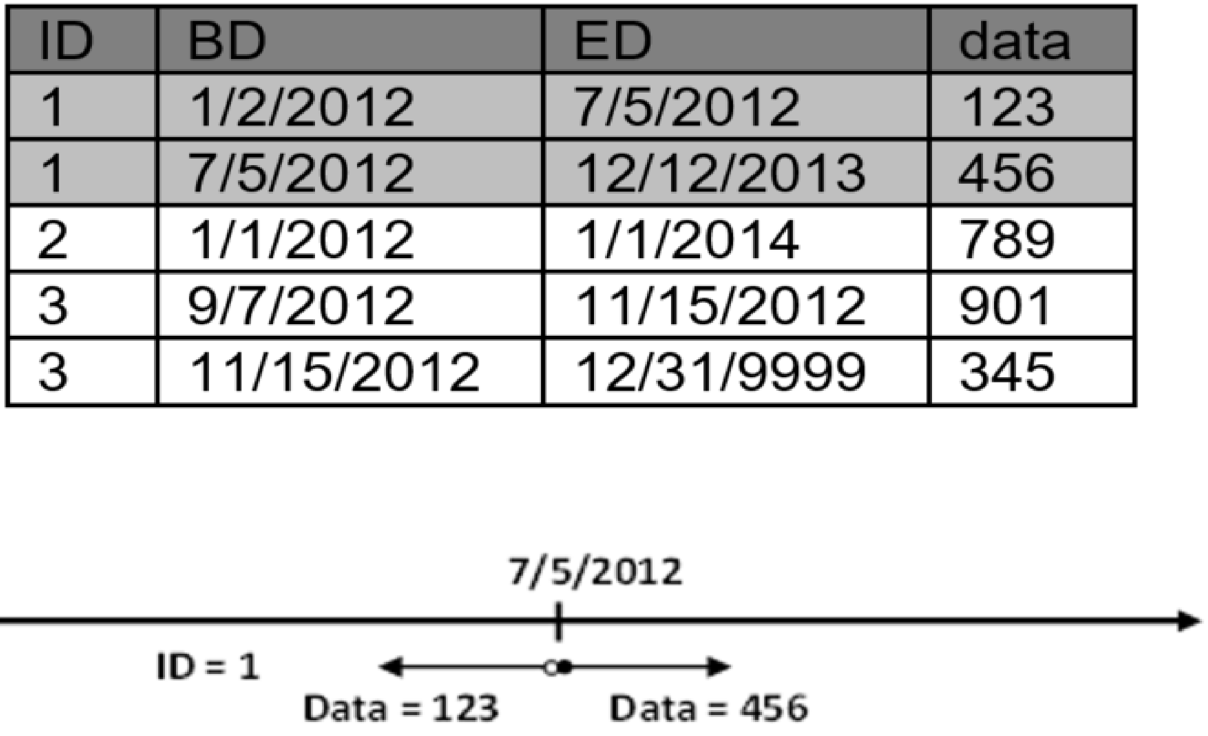
Principles of time interval management has been introduced in in the nineties of the 20th century. Also special datatype has been introduced delimiting begin (*BD*) and end (*ED*) date of the time interval – *period*. It was part of the concept, which was, however, to become temporal standard. Therefore, such type is not part of the SQL, nor specific dialect of individual database systems. It means, that interval definition must be user managed with regards on correctness – minimal interval length based on granularity, modelling method, correct position of the *BD* and *ED*. Furthermore, temporal data object can be defined by only one state in any time point. Such properties and activities should be monitored by period temporality core manager, however, it has not been approved resulting in necessity to define own criterions and conditions for each system.

Nowadays, time interval is defined mostly by two attributes replacing period data type. Thus, primary key of the object temporal table is extended by two attributes. Whereas primary key, nor any part of it cannot contain *NULL* values, time interval infinity cannot be modelled using *NULL* values. Naturally, begin date (*BD*) is still defined correctly, there is no necessity for non-strictness or non-precision. Vice versa, another aspect is used for end date (*ED*). Often, we do not know the time point of new update operation – in time of inserting data (new state), validity limit message in undefined (unknown). Also in systems, where data are produced regularly in defined time frame, there can be problems with interface input queues or communication methods. So, it cannot be strict. One of the possible solution is to exclude *ED* from the primary key, which is, although, possible, but they are commonly grouped together from the definition. Moreover, if there is *NULL* value delimiting validity, what does it mean? Can you be sure, that such time point has not occurred, yet? Can you be sure, that no newer event influenced such state sooner? No at all. Therefore, *NULL* values are really not suitable. Thus, we introduce *MaxValueTime* notation for dealing with such unlimited validity state management. Time interval itself can be modelled using two methods [4] [9]:

• closed-closed interval representation (*CC* representation),

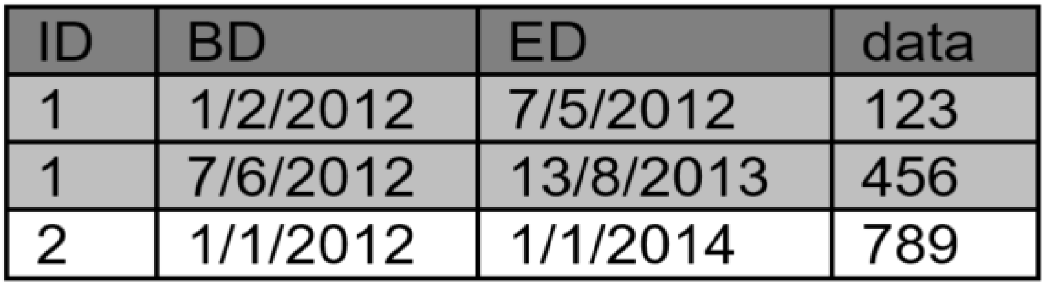
• closed-open interval representation (*CO* representation).

The difference between them is based on limiting time point of the interval. If using *CC* representation, the last time point is part of the interval, whereas *ED* of the *CO* representation expresses the first point, which can have another state delimited by the time. Significant advantage of using *CO* representation is offered, if there is necessity to transfer solution to another granularity. In this case, it is often useful or even inevitable to use more grained granularity. *CO* representation characteristics does not create undefined time spectrum, also called as time gaps [4] [5]. Transformation techniques and access methods are described in [1] [8] [9].



ID	BD	ED	data
1	1/2/2012	7/5/2012	123
1	7/5/2012	12/12/2013	456
2	1/1/2012	1/1/2014	789
3	9/7/2012	11/15/2012	901
3	11/15/2012	12/31/9999	345

Figure 2. Closed – open representation



ID	BD	ED	data
1	1/2/2012	7/5/2012	123
1	7/6/2012	13/8/2013	456
2	1/1/2012	1/1/2014	789

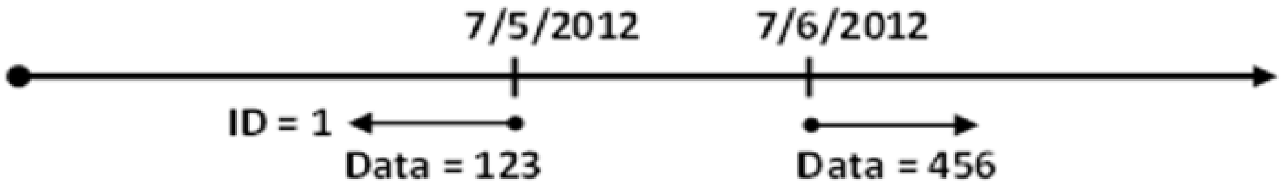


Figure 3. Closed – closed representation

3. MaxValueTime Notation

Special value for actual unlimited validity should be used to express, that the limitation is now unknown, but has not occurred, yet. This factor cannot be marked by the *NULL* value, because it is not possible to compare it with actual value (*sysdate*). Therefore, for such reasons, we introduce special notation for dealing with unlimited validity. In principle, it can be said, that it denotes the last possible timepoint value to be stored, respectively very high time value - *12-31-9999 (MM-DD-YYYY)* and time spectrum based on defined granularity. Thus, there is no problem with attribute value comparison using relational operations, to sort data. However, most significant factor is performance and access methods, which can be associated with this notation. In general, *MaxValueTime* notation is associated with *ED* and expresses "*until further notice*". Physically, *MaxValueTime* notation is used for storing data, not physical date, so there is no problem with consecutive interpretation.

4. Index Structure

Temporal data are characterized by huge data amount to be processed, evaluated and stored. Data analysis and retrieval must be performed with regards on effectivity and performance. Data are stored physically in data files accessed through interim layer – tablespaces. Each required block must be loaded into memory buffer cache before processing, which requires data resources and I/O operations and can be considered as important factor influencing performance - costs. Therefore, one of the main features affecting performance is delimited by access method reflected by using index structures. Temporal databases are state oriented with emphasis on evolution monitoring over the time. Getting states and individual changes in the *Select* statement forms the core of a major milestone of efficiency and speed of the system.

Oracle defines an index as an optional structure associated with a table or table cluster that can sometimes speed data access. By creating an index on one or more columns of a table, you gain the ability in some cases to retrieve a small set of randomly distributed rows from the table. Indexes are one of many means of reducing disk I/O [6] [10] [16].

The absence or presence of an index does not require a change in the wording of any SQL statement. An index is a fast access path to a single row of data. It affects only the speed of execution. Given a data value that has been indexed, the index points directly to the location of the rows containing that value. Database management system automatically maintains the created indexes – changes (*Insert, Delete, Update*) are automatically reflected into index structures. However, the presence of many indexes on a table degrades the performance because the database must also update the indexes [6] [17] [18].

There are several methods for accessing data, which can be used as a result of parsing optimizer decision. In this paper, we will mention the most important based on performance are following experiments:

• *Table Access Full*. This method sequentially accesses and loads all data blocks from the disc storage to the memory, which are under the *High Water Mark* (HWM). Whereas it must process all data blocks, it is also the slowest method for data retrieval. Moreover, it is used, when large portion of the table´s data is required (optimizer assumes, that accessing direct data would be easier and faster) or if accessed table is small consisting of few data blocks. In that case, *Table Access Full* method is used regardless the associated index.

• *Index Unique Scan* can be characterized by accessing data based on unique index, if only one value reflecting equality is used. It accesses no more than one data row.

• *Index Range Scan* is used, if suitable index is defined and condition is based on non-equality or equality in case of non-unique index.

• *Index Skip Scan* is special and new index access path type, which has been introduced in Oracle 9i version [10] [22]. It creates index tree, which node is index tree, too, but based on different leading attribute. Thanks to that, such method can process and locate data by skipping leading index column of the composite index.

These methods are based on locating particular data by using index. The result is direct data locator in the physical storage – *Rowid* (*Rowid* is pseudocolumn returning adress of the row consisting of these attributes – data object number, identifier of data file, identifier of data block and position of the row in the block). Then *Index Rowid method* can be used – data are located using obtained *Rowid*. Special type of the index approach is *Full Index Scan* and *Fast Full Index Scan*. These two methods are based on the fact, that all required data are part of the defined index, thus, no *Rowid* is necessary to be used. The difference between them is sorting. *Full Index Scan* method uses sorted data, whereas *Fast Full Index Scan* are not sorted in suitable way.

5. B+tree Index Type

The index structure of the B+tree is mostly used because it maintains the efficiency despite frequent changes of records (*Insert*, *Delete*, *Update*). B+tree index consists of a balanced tree in which each path from the root to the leaf has the same length [10] [22].

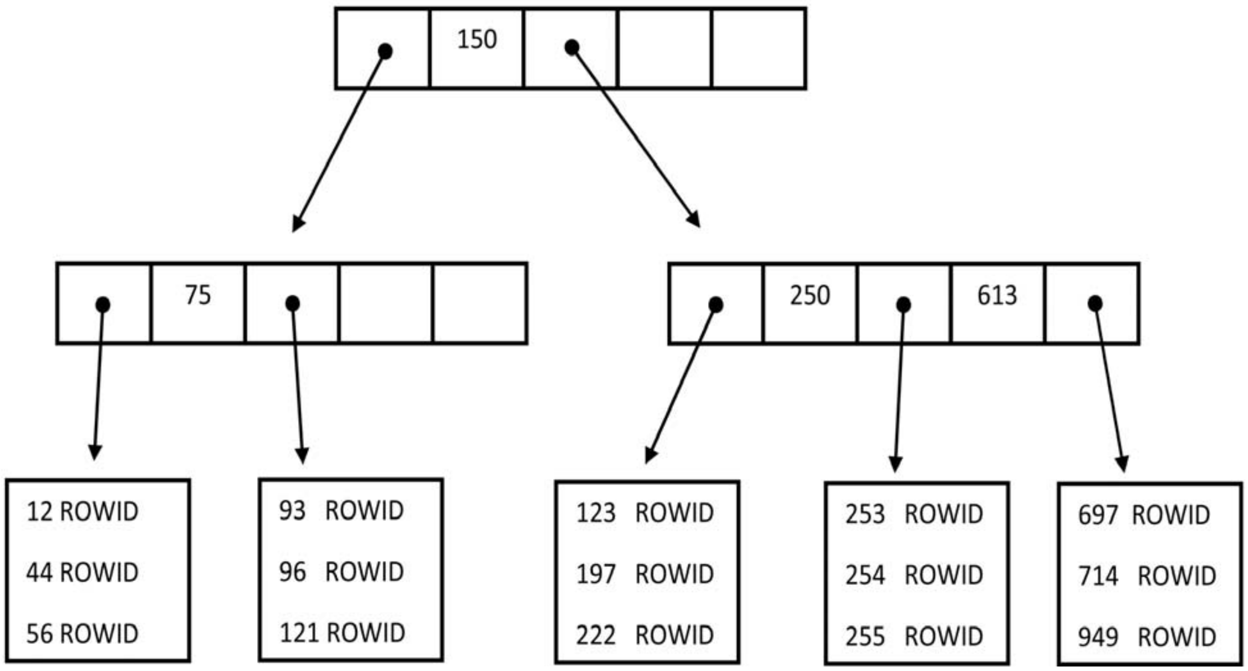


Figure 4. B-tree

In this structure, we distinguish three types of nodes - root, internal node and leaf node. Root and internal node contains pointers S_i and values K_i, the pointer S_i refers to nodes with lower values the corresponding value (K_i), pointer S_{i+1} references higher (or equal) values. Leaf nodes are directly connected to the file data (using pointers) [10].

B+tree (fig. 5) extends the concept of B-tree (fig. 4) by chaining nodes at leaf level, which allows faster data sorting. DBS Oracle uses the model of two-way linked list, which makes it possible to sort ascending and descending, too [10] [12].

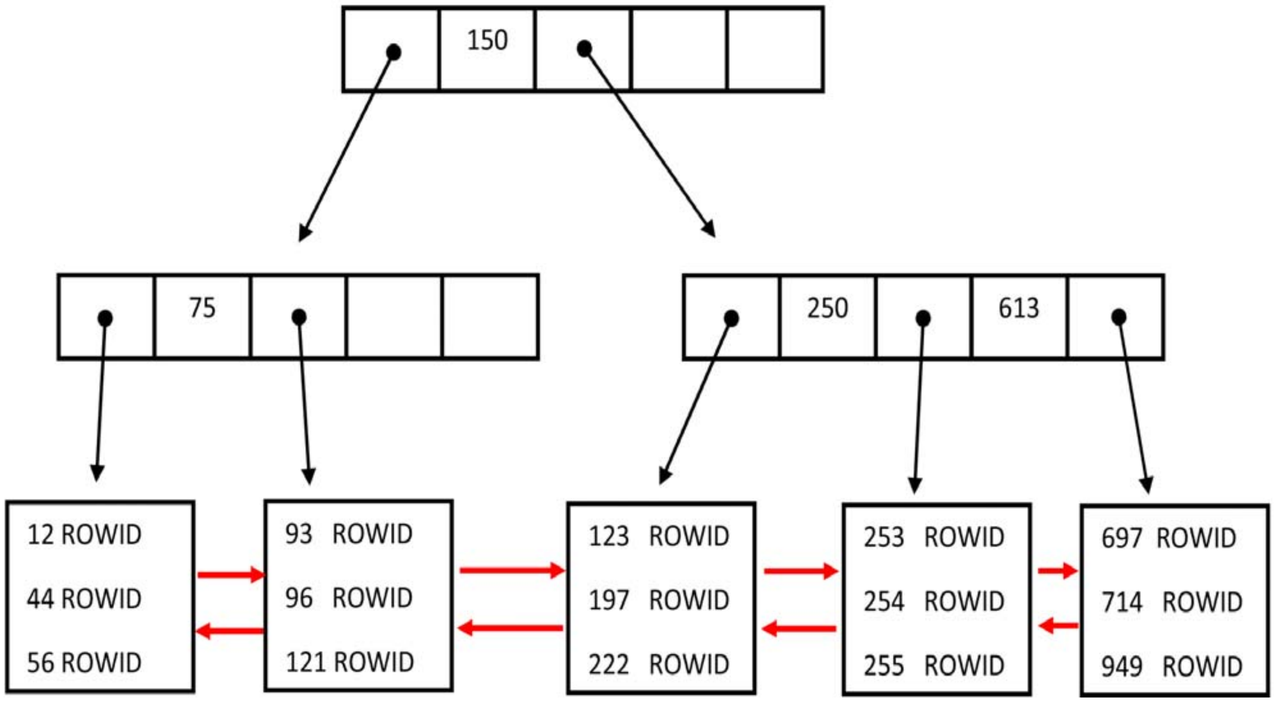


Figure 5. B+tree

Limitation of this approach is *NULL* value processing. Whereas such values cannot be compared using relational operations, they cannot be stored using this structure, which can result in significant performance degradation. Next chapters compares performance of managing undefined time limited states using existing *NULL* value approach and our proposed *MaxValueTime* and *UndefTypeValue* notation.

6. Experiment Environment

Our experiments and evaluations were performed using defined example table - *employee*. Fig. 6 shows the structure of the table. 50 departments were used, each consisting of 1000 employees, each of them was delimited by 20 different salaries over the time. Thus, total number of rows was one million. No primary key was defined, because of the environment properties and our direct opportunity for explicit index definition.

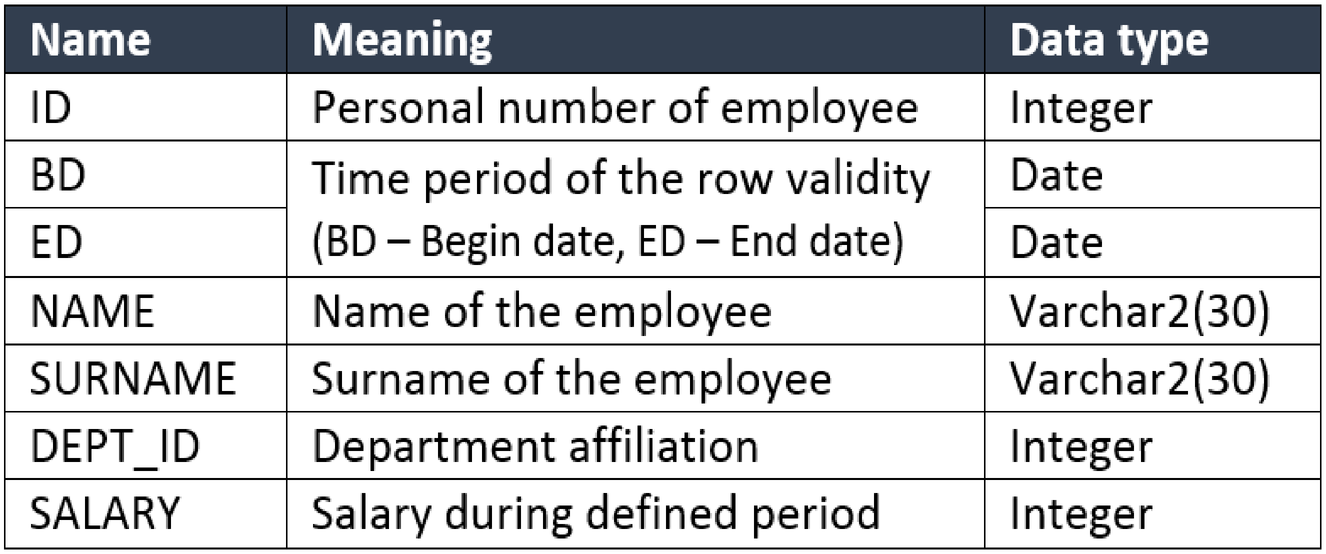
Experiment results were provided using Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production; PL/SQL Release 11.2.0.1.0 – Production. Parameters of used computer are: processor: Intel Xeon E5620; 2,4GHz (8 cores), operation memory: 16GB and HDD: 500GB.

Experiment results comparison was obtained using *autotracing*. These parameters were monitored:

• *access method (operation),*

• *CPU costs (in [%]),*

• *processing time (in [hh:mi:ss]).*



Name	Meaning	Data type
ID	Personal number of employee	Integer
BD	Time period of the row validity (BD – Begin date, ED – End date)	Date
ED		Date
NAME	Name of the employee	Varchar2(30)
SURNAME	Surname of the employee	Varchar2(30)
DEPT_ID	Department affiliation	Integer
SALARY	Salary during defined period	Integer

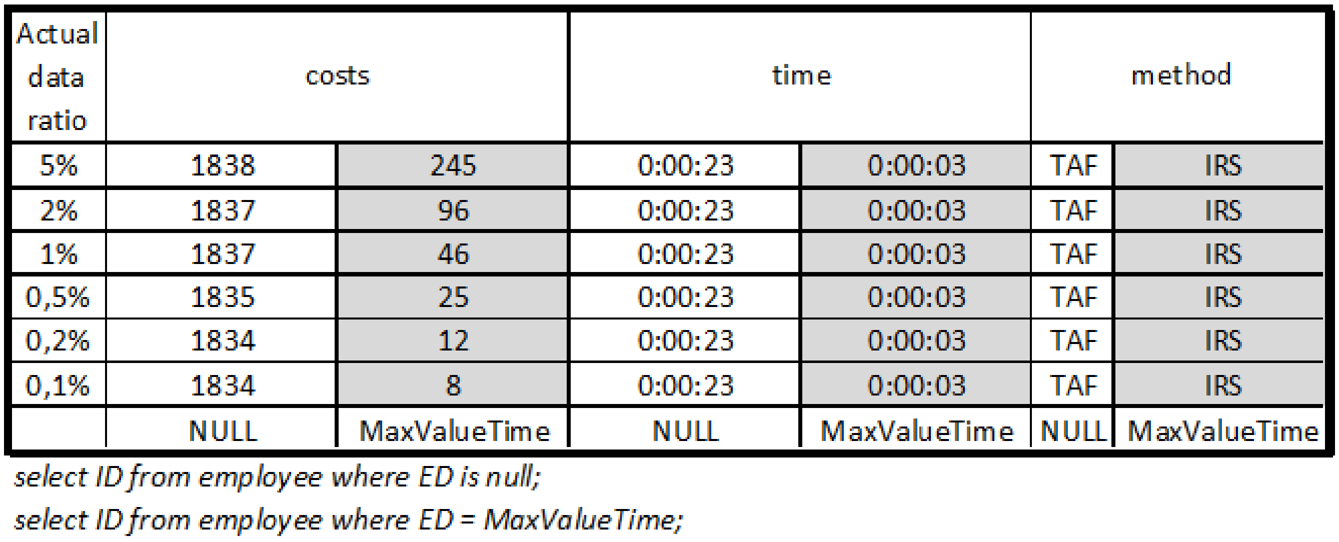
Figure 6. Employee table structure

7. Performance

To point out management of undefined values over the time, it is necessary to evaluate our proposed solution with existing approach to declare performance as well as define limitations. Whereas temporal characteristics requires each state to be defined by no more than one row, our defined environment limits the number of actual states to 50 000. In the following experiments, various number of actual states is used:

- 50 000 (5% of all table data),
- 20 000 (2% of all table data),
- 10 000 (1% of all table data),
- 5 000 (0,5% of all table data),
- 2 000 (0,2% of all table data),
- 1 000 (0,1% of all table data).

In the first phase, comparison of undefined time value denoted by *NULL* value in comparison with *MaxValueTime* notation is used. B+tree index based on attributes *ED*, *BD* and *ID* is created (in such defined order) – the reason and comparison of index types can be found in [12]. *Select* clause of the statement consists of only ID attribute, thus, all data can be obtained using index with no necessity for accessing data files in the physical storage. Fig. 7 shows the experiment results. As we can see, if *NULL* values are used, there is no other way, so *Table Access Full (TAF)* method must be used to avoid *NULL*s. If undefined value is modelled by *MaxValueTime* notation, all values are indexed, so *Index Range Scan (IRS)* with significant performance improvement can be used. Total costs and processing time reflect significant performance growth, too. If all data have actual non-limited value, 86,67% of costs is eliminated, which reflects 86,96% of processing time. With the reduction of the number of undefined values, the difference is even more strict – 99,56% of costs and 86,96%. As we can see, processing time does not depend on number of actual data ration. The reason is based on necessity of index loading into memory, which reflects the same time. On the other hand, total costs cover not only memory, but also other server resources, are eliminated with data dimension down tendency. Special category covers *Table Access Full* method, which must load and evaluate all data blocks of the table covered by *High Water Mark* (which does not transfer its value to lower segment). Our experiments use only new *Insert* statements execution, however, in commercial environment, also *Update* statements would be performed resulting in table segment fragmentation, thus table would contain more data blocks than necessary (some of them would not be fully occupied). The result would be again reduced performance of the *NULL* value management solution. Graphical result reporting is shown in fig. 8.



Actual data ratio	costs		time		method	
5%	1838	245	0:00:23	0:00:03	TAF	IRS
2%	1837	96	0:00:23	0:00:03	TAF	IRS
1%	1837	46	0:00:23	0:00:03	TAF	IRS
0,5%	1835	25	0:00:23	0:00:03	TAF	IRS
0,2%	1834	12	0:00:23	0:00:03	TAF	IRS
0,1%	1834	8	0:00:23	0:00:03	TAF	IRS
	NULL	MaxValueTime	NULL	MaxValueTime	NULL	MaxValueTime

select ID from employee where ED is null;
select ID from employee where ED = MaxValueTime;

Figure 7. Performance results – NULL, MaxValueTime

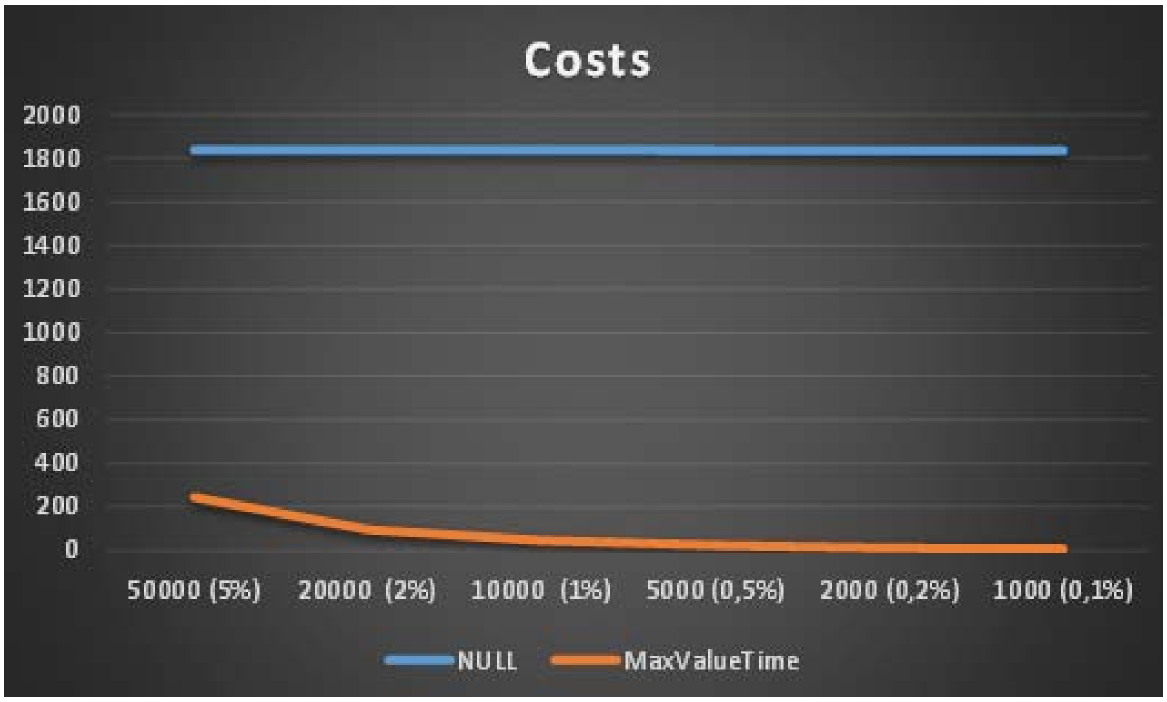
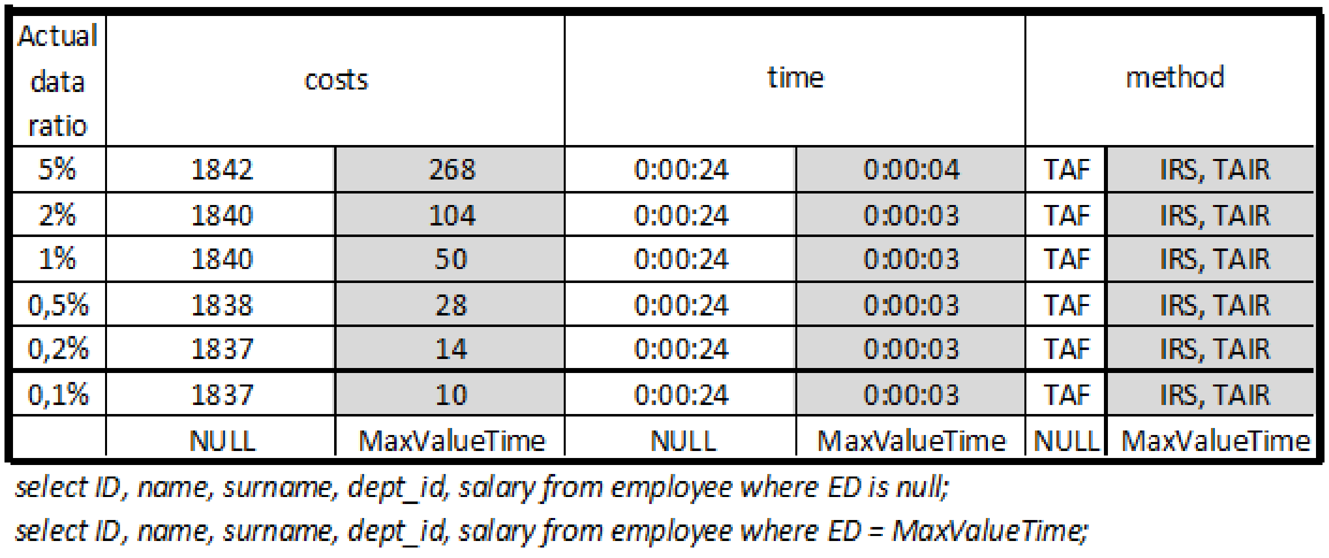


Figure 8. Costs– NULL, MaxValueTime

Second part deals with the extension of the *Select* clause. In this section, we require more attributes than are part of the index, therefore data must be accessed by *Rowids*, if index method is used. Again, *NULL* value performance is really worse. It does not express significantly different results in comparison with previous experiment. It always reflects all data blocks, which are loaded into memory. Data processing and transferring into result set from memory is negligible. *MaxValueTime* notation offers wide range of performance improvement possibilities. Also in this case, *Index Range Scan* method can be used, which must be, however, followed by loading particular data block to provide data, which are not part of the index. This method is called *Table Access by Index Rowid* (TAIR). It reflects the slowdown from 8% up to 25% in our environment. Average value of the added costs is 13,35%. Fig. 9 shows the results.



Actual data ratio	costs		time		method	
5%	1842	268	0:00:24	0:00:04	TAF	IRS, TAIR
2%	1840	104	0:00:24	0:00:03	TAF	IRS, TAIR
1%	1840	50	0:00:24	0:00:03	TAF	IRS, TAIR
0,5%	1838	28	0:00:24	0:00:03	TAF	IRS, TAIR
0,2%	1837	14	0:00:24	0:00:03	TAF	IRS, TAIR
0,1%	1837	10	0:00:24	0:00:03	TAF	IRS, TAIR
	NULL	MaxValueTime	NULL	MaxValueTime	NULL	MaxValueTime

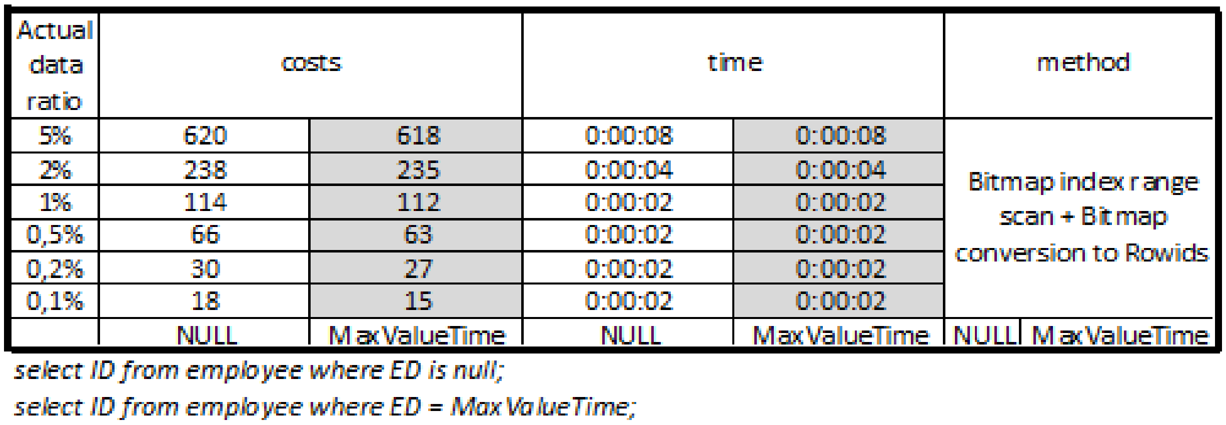
select ID, name, surname, dept_id, salary from employee where ED is null;

select ID, name, surname, dept_id, salary from employee where ED = MaxValueTime;

Figure 9. Costs– NULL, MaxValueTime

The last performance evaluation is this category is based on replacing existing *B+tree* index structure with *bitmap* index, which can manage *NULL* values. *Bitmap* index is advisable mostly for systems, which values are not changed at all, or the frequency is low. It is not, however, our case. Therefore, the task is to compare performance. The environment characteristics are the same, only *ID* attribute is selected. The total costs of the *NULL* processing are significantly lowered, whereas defined index can be used, which is also reflected by using *Bitmap Index Range Scan* and *Bitmap Conversion to Rowids* methods.

Processing costs based on *bitmap* index reflect improvement from 66% (5% of all data) up to 99% (0,1% of all data) in comparison with managing *NULL* values in B+tree index. Results are shown in fig. 10.



Actual data ratio	costs		time		method	
5%	620	618	0:00:08	0:00:08	Bitmap index range scan + Bitmap conversion to Rowids	
2%	238	235	0:00:04	0:00:04		
1%	114	112	0:00:02	0:00:02		
0,5%	66	63	0:00:02	0:00:02		
0,2%	30	27	0:00:02	0:00:02		
0,1%	18	15	0:00:02	0:00:02		
	NULL	MaxValueTime	NULL	MaxValueTime	NULL	MaxValueTime

select ID from employee where ED is null;

select ID from employee where ED = MaxValueTime;

Figure 10. Costs– NULL, MaxValueTime

Nevertheless the mentioned *bitmap* solution improvement, it is necessary to evaluate the comparison with our proposed *MaxValueTime* notation, which reflects better performance also for *bitmap* index. Specifically, for 5% of all data, processing, costs cover almost 100%, however, when data amount is reduced - 0,1% of all data, it reflects only 83% of total costs (*NULL* value processing using *bitmap* index is reference – 100%).

Comparison of the *bitmap* index using *NULL* values and *MaxValueTime* notation in *B+tree* index environment is the inevitable part of the proposed concept. In this part, we can see, that there is significant performance improvement, which is also reflected graphically in fig. 11:

• Blue colour expresses B+tree and *NULL* value processing.

• Orange colour expresses B+tree and *MaxValueTime* notation.

• Grey colour expresses *Bitmap* index and *NULL* value processing.

• Yellow colour expresses *Bitmap* index and *MaxValueTime* notation.

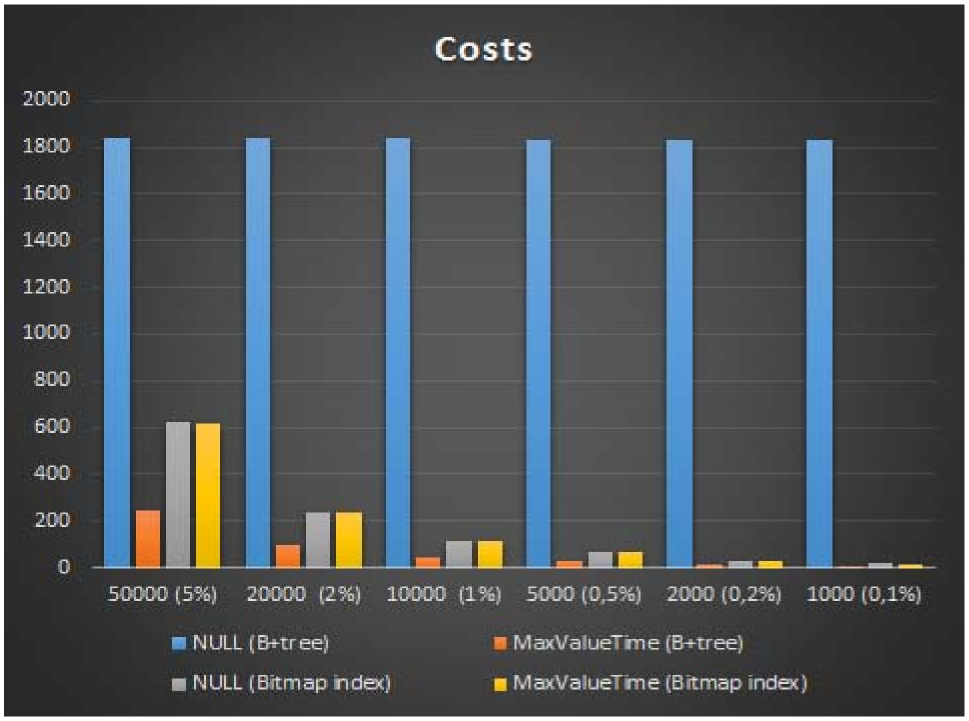
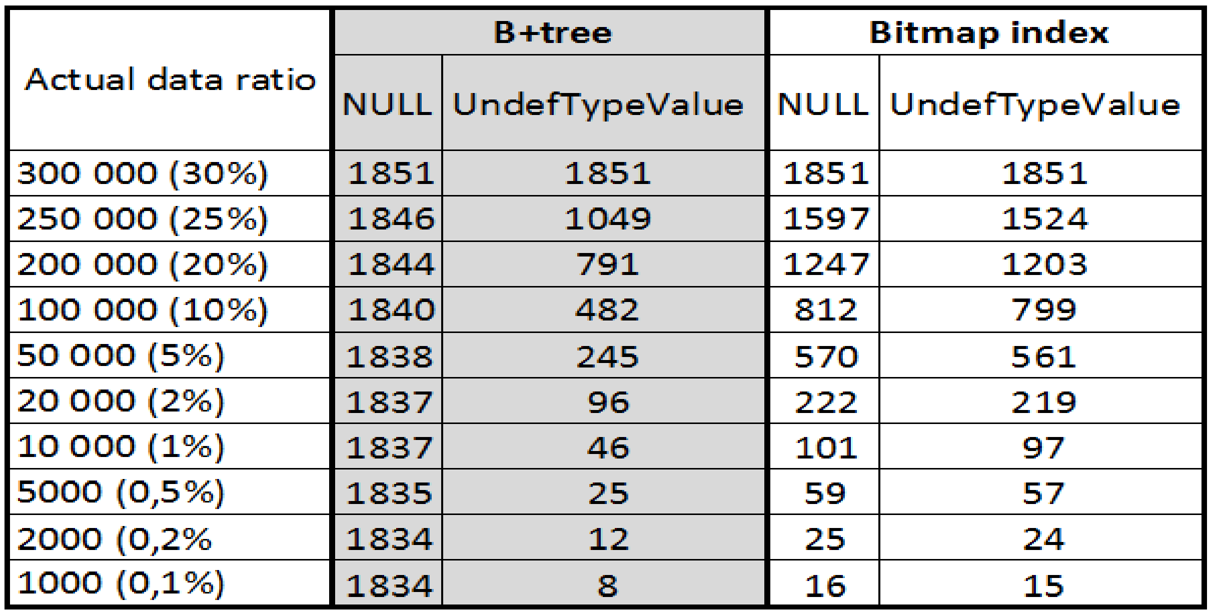


Figure 11. Costs– B+tree and Bitmap index



Actual data ratio	B+tree		Bitmap index	
	NULL	UndefTypeValue	NULL	UndefTypeValue
300 000 (30%)	1851	1851	1851	1851
250 000 (25%)	1846	1049	1597	1524
200 000 (20%)	1844	791	1247	1203
100 000 (10%)	1840	482	812	799
50 000 (5%)	1838	245	570	561
20 000 (2%)	1837	96	222	219
10 000 (1%)	1837	46	101	97
5000 (0,5%)	1835	25	59	57
2000 (0,2%	1834	12	25	24
1000 (0,1%)	1834	8	16	15

Figure 12. Costs– NULL, UndefTypeValue

Previous part of this section deals with temporal management in time attribute value processing – validity - with regards on undefined and non-bordered values. The aim of the second part is to compare performance of undefined value management, which, does not, however, have time characteristics. For these purposes, we will evaluate *salary* attribute in the same structure and the same environment properties. In this case, there is no ratio limitation, generally, all values can get *NULL* values. Thus, another solution, which avoids *NULL* values, can profit significantly. Therefore, we introduce *UndefTypeValue* notation, which references undefined value based on particular attribute datatype. For each SQL data type, special notation is proposed. If our *UndefTypeValue* constraint is used, index access method is used up to 25% data ratio. Then, optimizer predicts, that accessing all data blocks will be easier for processing. In this case, we will evaluate only processing costs. Results are in fig. 12 and fig. 13.

Stricter results would be reflected in attribute oriented temporal approach.

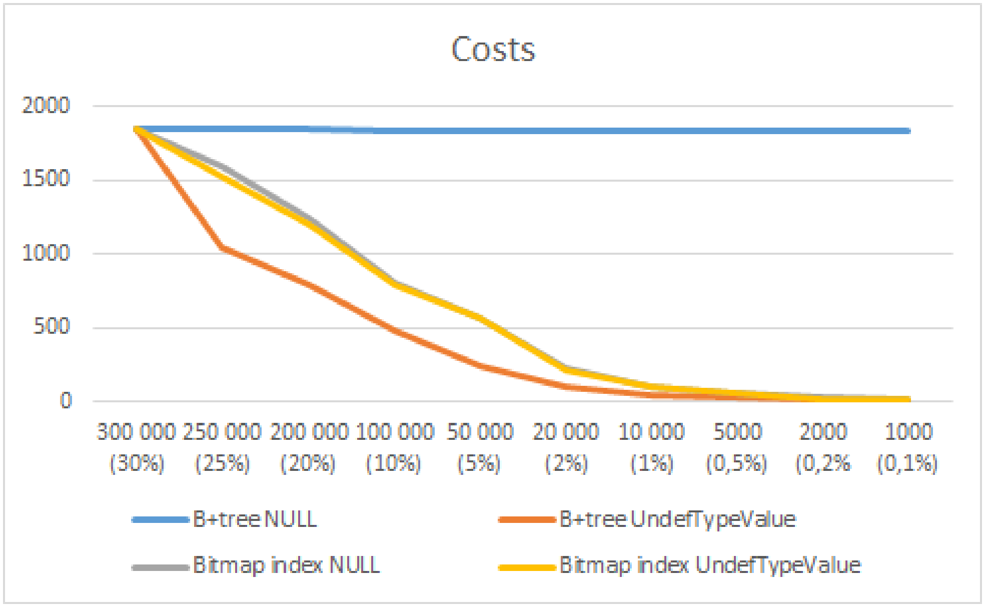


Figure 13. Costs– NULL, UndefTypeValue

8. Attribute Oriented Temporal Model

Attribute oriented temporal model was firstly introduced in 2013 and is based on shifting previous solution to the attribute granularity. Solution consists of three levels ensuring existing applications managing only conventional approach to be used without application code compiling necessity. Core of the system is temporal model, which reflects all changes and points to the historical, actual, and also future valid value. Thanks to that, there is no duplicate values in the system. It consists of these attributes [13] [14] [15]:

• *ID_change* – got using sequence and trigger – primary key of the table.

• *ID_previous_change* – references the last change of an object identified by *ID*. This attribute can also have *NULL* value that means, the data have not been updated yet, so the data were inserted for the first time in past and are still actual.

• *ID_tab* – references the table, record of which has been processed by DML statement *(Insert, Delete, Update, Restore)*.

• *ID_orig* – carries the information about the identifier of the row that has been changed.

• *ID_column* – holds the information about the changed attribute (each temporal attribute has defined value for the referencing).

• *Data_type* – defines the data type of the changed attribute:

  • *C = char / varchar*

  • *N = numeric values (real, integer, …)*

- *D = date*
- *T = timestamp*

This model can be also extended by the definition of the other data types like binary objects.

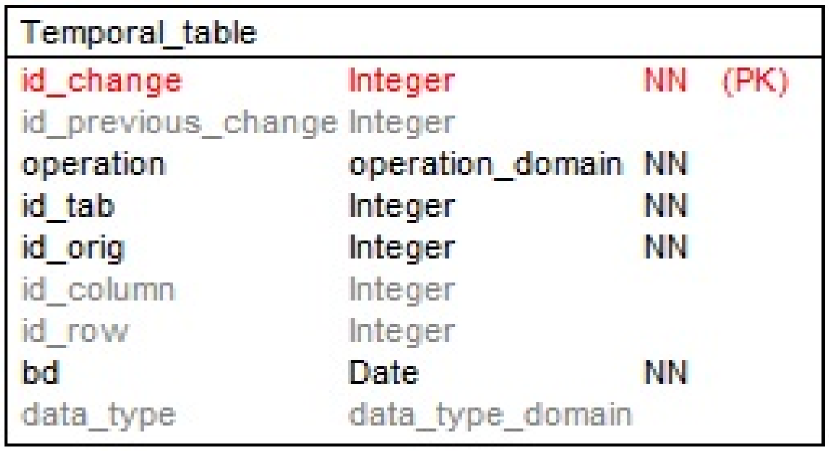
• *ID_row* – references to the old value of attribute (if the *DML* statement was *Update*). Only *Update* statement of temporal column sets not *NULL* value.

• *Operation* – determines the provided operation:

- *I = Insert*
- *D = Delete*
- *U = Update*
- *R = Restore*

The principles and usage of proposed operations are defined in the part of this paper. *Restore* operation provides validity restoration after the sensor or data distribution failure.

• *BD* – the begin date of the new state validity of an object.



Temporal_table			
id_change	Integer	NN	(PK)
id_previous_change	Integer		
operation	operation_domain	NN	
id_tab	Integer	NN	
id_orig	Integer	NN	
id_column	Integer		
id_row	Integer		
bd	Date	NN	
data_type	data_type_domain		

Figure 7. Temporal layer structure

Despite of the performance improvements, such solution is still not optimal, whereas each change of the attribute automatically means performing *Insert* statement into temporal system model, which is also the bottleneck of the system. In many systems, some attributes and their changes are correlated and synchronized. If multiple values are always changed at the same time, why do we need multiple *Insert* statements with the same values delimiting time? Are there duplicates, aren´t they? Sure, there is problem. Solution resides in creating attribute groups. Temporal model then is not delimited only by attribute itself, but it can reference also attribute group. Principle of such solution is described in [14].

Extension of temporality is based on reliability management [11] and network optimization, simulation [19] [20] covered by security management [13].

9. Conclusion

Temporal enhancement of the database systems brings the opportunity to store, manage and evaluate data during the whole lifecycle of the objects. At the same time, it requires extension of the processing paradigm, which is based on conventional, non-temporal approach. Data management efficiency as well as performance is one of the key part of our research. Object level temporal approach, which is based on object granularity itself, does not provide sufficient power for dealing with objects, which attribute values are not updated at the same time, which delimits the performance, but also storing capabilities. Therefore, we propose column level temporal architecture, which is based on attribute granularity and offers significant performance improvements. With this approach, sensorial data can be provided using any granularity and frequency of changes, temporal management layer provides robust solution to reflect performance.

However, many times, it can happen, that there is no relevant value to be stored, it is certain, that actual value is not reliable. In object level temporal model, the whole state is considered as unknown, whereas attribute oriented approach deals with such validity and reliability problem delimited by unknown value separately. In this paper, we define limiting factors of managing undefined values in temporal environment with regards on index approach performance. As we can see in the experiments, there can be significant improvement, if no NULL values are stored regardless the time delimiting validity in any temporal sphere or the attribute value itself. Therefore, two notations are introduced and evaluated. MaxValueTime is associated with time attributes, which mostly reflect validity. From the temporal definition, it can be used only for right border of the interval – ED. UndefTypeValue is generalization of the approach and i sused for any attribute value, which does not reflect actual or correct value. It is developed for each data type, which can be structured using undefined values.

In the future, we would like to highlight grouping factors in attribute oriented temporal approach with emphasis on undefined value in group, but the rest part of the group would remain defined and consistent. It can also reflect dependencies to evaluate and replace unknown or incorrectly provided values by adding additional layer providing sophisticated solution with prediction and complex evaluation techniques.

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 the result of the project implementation *Center of translational medicine*, ITMS 26220220021 supported by the Research & Development Operational Programme funded by the ERDF.

Európska únia

Agentúra
Ministerstva školstva, vedy, výskumu a športu SR
pre štrukturálne fondy EÚ

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

References

[1] Ahsan, K., Vijay. P. (2014). Temporal Databases: Information Systems, Booktango.

[2] Avilés. G. (2016). Spatio-temporal modeling of financial maps from a joint multidimensional scaling-geostatistical perspective, *In Expert Systems with Applications*. V. 60, p. 280-293.

[3] Behling. R. (2016). Derivation of long-term spatiotemporal lanslide activity – a multisensor time species approach, *Remote Sensing of Environment*, 136, p. 88-104.

[4] Date, C. J. (2006). Date on Database, Apress.

[5] Date, C. J. (2007). Logic and Databases – The Roots of Relational Theory, Trafford Publishing.

[6] Feueuerstein, S. (2014). Oracle PL/SQL Programming, O'Reilly.

[7] Huey, P. (2014). Oracle Database Security Guide, Oracle Press.

[8] Johnston, T. (2014). Bi-temporal data – Theory and Practice, Morgan Kaufmann.

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

[10] Kuhn, D. (2012). Expert Indexing in Oracle Database 11g, Apress.

[11] Kvassay, M., Zaitseva, E., Kostolny, J., Levashenko, V. (2015). Importance analysis of multi-state systems based on integrated direct partial logic derivatives, *In:* 2015 International Conference on Information and Digital Technologies, p. 183–195.

[12] Kvet, M. Temporal data approach performance, APSAC 2015, p. 75 – 83.

[13] Kvet, M., Matiasko, K. (2015). Temporal Context Manager. SDOT •ilina, p. 93-103.

[14] Kvet, M., Matiasko, K. (2014). Transaction Management. CISTI, Barcelona, p.868-873.

[15] Kvet, M., Matiaško, K., Kvet, M. (2014). Complex time management in databases, *Central European Journal of Computer Science*, 4 (4) 269-284.

[16] Matiaško, K., Vajsová, M., Zábovský, M., Chochlík, M.(2008). Database systems. EDIS.

[17] Niemiec, R. (2014). Oracle Query Tuning, Oracle Press.

[18] Rood, R. (2012). Oracle Advanced PL/SQL Developer Professional Guide, Packt Publishers.

[19] Simsion, G., Witt, G. (2005). Data Modeling Essentials, Morgan Kaufmann.

[20] Suarez, E. (2016). Reconstruction of Neural Acrivity from EEG Data Using Spatiotemporal Constraints, *International Journal of Neural Systems*. Vol. 26.

[21] Tuzhilin. A. (2016). Using Temporal Logic and Datalog to Query Databases Evolving in Time, Forgotten Books.

[22] Watson, J. (2008). OCA Oracle Database 11g Administration, Oracle Press.