https://doi.org/10.26552/com.C.2014.1.97-104

Michal Kvet - Karol Matiasko *

COLUMN LEVEL UNI-TEMPORAL DATA

The basic paradigm of conventional database systems is based on the current valid data processing. However, today's database systems should provide also management for historical and future valid data. Standard temporal model, using object level, stores the whole object data after the insert or update operation, although only some attributes changed their values. This paper deals with the principle of temporal data modelling based on the column level, not the whole object, describes the structure and principles of required methods, procedures, functions and triggers to provide functionality of the system. All designed and implemented solutions are compared with the existing solutions.

Keywords: *Conventional database, temporal database, column level temporal data, fully temporal model.*

1. Introduction

Database systems are one of the most important parts of the information technology. Almost certainly it can be said that a database system is the basic part, the root of any information system. The development of data processing has brought the need for modelling and accessing large structures based on the simplicity, reliability and speed of the system [1].

Most of the data in the database represent the current state, the data valid at this point. Properties and states of the objects evolve over the time, become invalid and are replaced by new ones. Once the state is changed, the corresponding data are updated in the database and it still contains only the current valid data. However, history management is very important in systems processing sensitive data; incorrect change would cause a great harm or in the systems requiring the possibility of restoring the previous states of the database. Therefore, it is necessary to store not only the current state, but also the previous states and progress. It can also help us to optimize processes or to make further decisions.

Historical data were saved using log files and archives in the recent past. Thus, the historical data could be obtained, but it is a complicated process, these data are in the raw form and handling them was difficult, lasted too much time. In addition, management requires quick and reliable access to data defined by any time point, but also getting information about the changes of the attributes in the future without significant time delays. However, the future valid data cannot be processed using mentioned methods at all.

The main disadvantage is the need of the administrator´s intervention (operation manager). An administrator must manage not only the running applications but also requirements for accessing historical backups. Decisions are based on historical data and the progress, so it was necessary to load historical backups to get the database snapshot at the historical time point. Operational decisions could not be based on the historical data because of the time consumption (sometimes even days to load all needed snapshots). In addition, the granularity of the data is still growing, so number of backup is above the acceptable level.

Nowadays, historical data management is easier than in the past, requires less processing time, but there is still need to make significant progress in temporal database processing research to create a complex module allowing to run existing applications without modifying source code and settings [2].

When managing temporal data, two aspects are considered to be most important. The aim is to provide the user easy and quickly manageable methods. The second criterion is the speed. We require the adequate processing time when managing historical and also future valid data. Last but not least, the requirement is based on the complex size of the database.

2. State of the art

Temporal databases define a new paradigm for selecting one or more rows based on the specified criteria, for projecting of one or more columns to the output sets and for joining the tables by

* **Michal Kvet, Karol Matiasko**
Department of Informatics, Faculty of Management Science and Informatics, University of Zilina, Slovakia
E-mail: Michal.Kvet@fri.uniza.sk

specifying relationship criteria. Rows with the different values of the primary key (*PK*) can represent one object at different times. Transactions for inserting, updating and deleting the rows must, therefore, specify not only the object itself, but also the processed period. If the valid time of the object is defined by a time interval, the transaction must include a time period - 2 time point values - begin and end timestamps (or other data structure based on the granularity, like the date). This means that the update query does not cause only update of existing data, but also insert of the new row based on the validity intervals [3, 4 and 5].

Row in a relational database table can be defined in three different ways using time (Fig. 1). *ID* is a unique identifier; *PK* refers to a primary key. *BD* and *ED1* is a pair of columns defining the beginning and end value of the period - validity, *BD2* and *ED2* defines the second time interval – transaction time. The first model does not use time for definition at all, it cannot provide management for non-current data in the main structure. This is a standard model used today, called the conventional model. The primary key is defined by the attribute *ID* (can be composite). All not key attributes in the table, regardless of their number, are merged into a common block called *data* [3, 4 and 6]. The uni-temporal system uses the composite primary key – object identifier and the time interval defining the validity state characterizing the row.

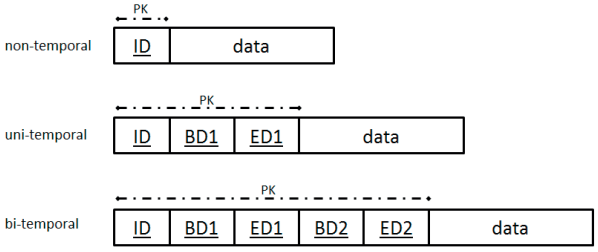

Fig. 1 Conventional and temporal table

This uni-temporal structure model thus allows defining the historical and also future valid data. However, the important factor for the time consumption and manipulation is the structure and its representation. Not all historical data are needed, if some attribute change monitoring is not important to us, we do not need to store it. In addition, a lot of duplicities are produced due to storing the whole rows, although only one attribute is going to be updated [4, 7 and 8].

Figure 2 shows the principle of data storing using the uni-temporal model. The interval is modelled using closed-open representation, which is, nowadays, the most often used method for time interval modelling. The other systems with open left side representation have problem with the definition of the begin validity time point. This method in comparison with the closed-closed representation does not have problem with changing granularity to smaller and does not need to update the table data to remove the undefined states. Ways of the modelling time intervals with the characteristics, features and principles for usage are described in [8].

ID	BD1	ED1	data
1	September 2012	September 2013	data1
2	January 2013	November 2014	data2
1	September 2013	December 2014	data11

Fig. 2 Uni-temporal table

The principle and importance of the bi-temporal data modelling is described in [8 and 9].

3. Uni-temporal model based on the begin time of the validity

Special type of uni-temporal system is a solution that contains only one time attribute that is part of the primary key. This

Tab_1_BD

ID	BD	data
1	Sep 2013	data
1	Dec 2013	data

1

Tab_1_BD_ED – *closed-closed interval*

ID	BD	ED	data
1	Sep 2013	Nov 2013	data
1	Dec 2013	*Dec 9999*	data

2

Tab_1_BD_ED – *closed-open interval*

ID	BD	ED	data
1	Sep 2013	Dec 2013	data
1	Dec 2013	*Dec 9999*	data

3

Fig. 3 Types of uni-temporal table modelling

means that any change of the corresponding object determines the validity of the prior state. The following Fig. 3 shows the representation of such a model, as well as corresponding standard uni-temporal system. The first part of the figure consists only of the definition of the begin time of the validity. The second one is a standard model with the closed-closed representation of the interval; the last consists of the model based on the closed-open representation.

The mentioned solution seems to be easy, but it does not address the fundamental problem of undefined states, the time intervals during which the state of the object is partially or completely undefined. We cannot easily replace the previous value with *NULL* value, because it can have special denotation. In addition, some attributes cannot have *NULL* value; it can be limited by the definition of the attribute column. The problem, however occurs, if even one attribute value is changed to the undefined state. The whole state of the object must be denoted as undefined or incorrect. It must be, therefore, possible to distinguish the condition of correctness and completeness of the object state. The easiest way is to delete this object from the database, because it cannot be referenced and even used for statistics and other manipulation. However, it would be necessary to remove the complete image of the object; the end date of the validity (new state determines the validity of the previous one) would be incorrect. If the state is no longer valid, history will not be stored. Thus, it will cause the temporal system degradation,. Object snapshots and images would disappear during the time. The question is easy, how to manage incorrect or incomplete states? Imagine the problem that only one attribute is unknown, for example, in industry caused by broken sensor or damaged cables.

One of the solutions is to add a flag representing the condition of the object state correctness. However, the problem is not solved completely. If one attribute value fails, the globalstate of the object is undefined. This is also the reason for defining the system based on the column level, where the whole state is a composition of the states of the temporal attributes.

4. Temporal column updating

Transformation of the conventional model to uni-temporal model on object level requires adding one or two attributes to extend the model with time validity. If there is a need to update the attribute, the whole state is modified and new one is inserted, thus, some attribute values are only copied to the new state because the values have not been changed. Moreover, some attributes do not need to be temporal, it is not necessary to monitor them or simply, they do not change their values over time. These values are still copied to the next image of the object. One of the solutions is based on a division of the original table. The first part contains non-temporal attributes, it is a classical conventional table, the second one is temporal, but it consists only of temporal columns – columns, which must be monitored and changes must be stored (Fig. 4). However, the problem is solved only partially, there is still problem with temporal columns which do not change their values in the time of update. This figure also shows the inefficiency of this system; if the number of temporal columns is high, the problem is more significant. Therefore, new system manipulating attributes not at object level must be developed.

Solution for temporal management should be universal, not only in terms of usability in practice, but also in terms of

ID	BD	$data_1$	$data_2$	...	$data_n$	non-temporal data
1	May 2012	A	B	C	X	data
1	Aug 2012	A_1	B	C	X	data
1	June 2013	A_1	B_1	C	X	data
1	Aug 2013	A_1	B_1	C_1	X_1	data

ID	BD	$data_1$	$data_2$	...	$data_n$
1	May 2012	A	B	C	X
1	Aug 2012	A_1	B	C	X
1	June 2013	A_1	B_1	C	X
1	Aug 2013	A_1	B_1	C_1	X_1

ID	non-temporal data
1	data
1	data
1	data
1	data

Fig. 4 Temporal table with non-temporal attributes

independence from the used database system. Creating new structures at the core level of the database system is, therefore, not appropriate. The basis of the proposed solution is to use existing resources and their combinations to create temporal solution. Moreover, it should be possible to be adapted into existing applications without the need of changing application programs.

The following Fig. 5 shows the developed and implemented structure. The existing program can continue to operate without any changes. The main part is to manage the table containing information about the changes of temporal columns. A column, whose changes need to be monitored, is temporal. If the value is changed, information about the update is stored in the developed

Fig. 5 Column level temporal system

temporal table and historical value is inserted into to the table containing historical values. Each temporal column has its own historical table.

The temporal table consists of the following attributes [7, 9 and 10] – see also Fig. 5:

- ID change
- *ID previous change* – references the last change of an object identified by *ID*. This attribute can also have *NULL* value which means the data have not been updated yet, so the data were inserted for the first time in past and are still current.
- *ID_tab* – references the table, record of which has been processed by DML statement (*INSERT*, *DELETE*, *UPDATE*).
- *ID_orig* - carries the information about the identifier of the row that has been changed.
- *ID_column*, *ID_row* – hold the referential information to the old value of attribute (if the DML statement was *UPDATE*). Only update statement of temporal column sets not null value.
- *BD* – the begin date of the new state of the object.

Figure 5 also shows the data representation and manipulation. The principles of the DML operations and related triggers source codes can be found in [7].

5. Future valid data definition

Data processing and data management valid in the future is also the requirement to the temporal system. Standard uni-temporal model does not have this problem; new data are inserted into the table which contains all the data about the objects during their life-cycle. The structure in Fig. 5 also allows managing future valid data. One of the easiest ways is based on the functionality *Job*, which allows you to plan data modelling operation to any time point in the future. It ensures the automatic update of the object state at defined time (*DBMS_SCHEDULER.CREATE_JOB*). Planning this event requires several parameters which are described in Fig. 6.

Create_job	
Job_name	Identifier of the job
Job_type	Type of the job (*plsql_block, stored_procedure, executable*)
Job_action	Statement
Start_date	Planned time to run the job
Repeat_interval	How often or when the job should be started
Enabled	The state of the job

Fig. 6 Job parameters

Information about the scheduled jobs is stored in the future table for each table consisting of one or more temporal columns. It has the same structure as the related conventional table, but it is extended by the operation (insert, delete, and update) and also by the identifier of the job. If the *Job* is executed, it provides deleting corresponding data from the future table, so this table contains only planned, but not executed jobs (Fig. 7).

Tab1_future			
job_name	Varchar2(10)	NN	(PK)
id_orig	Integer	NN	
operation	operation	NN	
x	Varchar2(30)		
y	Varchar2(30)		
z	Varchar2(30)		

Fig. 7 Structure of the future table

The method for executing the operation consists of the three parts (complete model is shown in Fig. 9):

- Productive table update *(insert, delete or update operation)* – Tab 1.
- *Insert into temporal_table* – it is provided using trigger.
- *Delete from the future table* – this table consists only of the planned, but not executed jobs.

However, there is much more principal complication of the future updates. The first problem is the process of dropping the related jobs if there is the incorrectly planned job – the values of the planned job are not correct, valid or simply, values do not correspond the future reality. Another problem is connected with the necessity of the jobs dropping based on the earlier update.. Suppose that there is a planned update of the object, but the system requires the earlier update (*T2*) of this object. The later planned (*T1*) job should be dropped. Another example is the need for replanning the job to another time.

7. Planned jobs management

Data about the planned *Jobs* can be found in the database system resources and tables. However, these data are raw and difficult to manipulate directly. Moreover, implemented system must dynamically react to any situation and request. Therefore, information about the planned jobs is stored in the related future tables (Figs. 7 and 9). Our implemented systems based on the column level were compared with each other and this presented model seems to be the most efficient [11]. Future table consists of the planned, but not executed jobs. However, planning the new job requires one more action, which has not been mentioned about yet, but it is expressed in Fig. 8. - the management of the jobs. If there is a request to change the object state (*insert*, *update*, *delete* operation), the system must search for any scheduled jobs based on this object. If there are some, the user is notified and must choose – retention or cancellation of the jobs - procedure ***drop_job_proc***.

```
create or replace procedure Job_proc(ID integer)
as
  cursor job_cur IS
    select job_name, id_orig, operation from Tab1_future where id_orig=ID;
    v_job_name varchar2(10);
begin
  open job_cur;
  loop
    fetch job_cur INTO v_job_name;
      if (job_cur%found) then drop_job_proc(v_job_name);
        else exit;
      end if;
  end loop;
end;
/
```

Fig. 8 Job management procedure

A complete structure of the fully temporal model can be seen in Fig. 9. The model consists of the classical non-temporal tables. Future data are stored in the future tables up to the execution or cancelling the process. The temporal table consists of the reference to historical values, so any state during the life-cycle can be reconstructed.

8. Experiments

The basis for the development of the new system is the performance compared with the existing systems to declare the characteristics, properties and limitations of the system. Our experiments and evaluations are based on the processing time, which, in our opinion, best represents the quality of the model. The second part contains the size of the model and related number of supplementary tables and structures. Figure 10 shows the experiment results based on the three models. The first model is the uni-temporal solution based on the object state management. The second one does not include the management for future valid data (Fig. 5). The last model is a complex fully temporal model with all the implemented structures and methods (Fig. 9).

ID2	X	Y	Z
A	X_A	Y_A	Z_A
B	X_B	Y_B	Z_B
C	X_C	Y_C	Z_C

Tab 1

ID2	Operation (I, U, D)	X	Y	Z	job_id
A	I	X_A	Y_A	Z_A	job_1
B	U	X_B	Y_B	Z_B	Job_2
C	D	X_C	Y_C	Z_C	Job_3

Tab 1 FUTURE

Temporal_table

ID_change PK	ID_ previous_change	ID_tab NN	ID_orig NN	ID_column	ID_row	BD NN
1	NULL	1	A	NULL	NULL	11-1-2012
2	NULL	2	E	NULL	NULL	11-2-2012
3	NULL	2	F	NULL	NULL	11-3-2012
4	1	1	A	Z	11	11-4-2012
5	4	1	A	Z	12	11-5-2012
6	2	2	E	„street“	45	11-6-2012
7	6	2	E	„post_code“	37	11-7-2012
8	1	-1	A	NULL	NULL	11-8-2012
9	7	-2	E	NULL	NULL	11-9-2012

ID	Value
...	...
44	Školská
45	Mostná

Fig. 9 Fully temporal model

	Uni-temporal system	Temporal table (historical data processing)	Temporal table (full – job info in future tables)
	MODEL 1	*MODEL 2*	*MODEL 3*
Type (level)	*Object*	*Column*	*Column*
Number of temporal columns	all	A <= all	A <= all
Number of conventional tables (containing temporal columns)	0	B	B
Number of temporal tables	B	1 (*developed*)	1 (*developed*)
Number of future tables	0	0	B
Number of historical tables	0	A	A
Size of the DB (kB)	41 235	18 093	18650
Time to get all actual data (current snapshot) (ms)	3921	931	931
Time to get all data during the life-cycle of one object (ms)	721	460	480

Fig. 10 Experiment results

The total number of records in the main structure is 100 000. The experiments were provided using the Oracle 11g database system. The experiment results can be divided into two categories which can be processed separately.

The first category evaluates temporal data. The second processed model deals with the temporal data, but only with historical data, which means, that there is no future table and possibility to manage these data using *Job*s. The third model can be considered as the extension of the second model using future valid data management.

The overall slowdown of the fully temporal structure (model 3) in comparison with model 2 is:

- **Size:** 3.08%.
- **Time to get current snapshot:** 0%.
- **Time to get all data during the life-cycle of one object:** 4.35%.

These results can be considered satisfactory. The increased requirement for the size of the structure is caused by the fully temporal system. The model 2 returns historical and current valid data, whereas the model 3 returns also future valid data.

The second category of experiments compares the temporal models based on the level. The first model is deals with the uni-temporal system on the object level. Each update causes the update of the whole object regardless of the number of the updated columns. All of the attributes in the application must be temporal. The model 3, which was developed by us, is based on the attribute – column level. Each state is, therefore, a merge of the data attribute values. Moreover, when some attribute data are not valid or are unknown, these columns are not processed, but the whole object can be considered valid or partially valid.

The overall acceleration of the developed system (model 3) in comparison with the standard uni-temporal system (model 1) is:

- **Size:** 54.77%.
- **Time to get current snapshot:** 76.25%.
- **Time to get all data during the life-cycle of one object:** 33.43%.

The developed system provides significantly better performance rate.

Overall results are influenced by the real number of temporal columns in comparison with the number of conventional attributes. However, in the most cases, it is not necessary to model systems with all temporal attributes..In the experiments, we used the system with one conventional and two temporal columns.

9. Conclusion

An object in the conventional database is represented by one row which expresses the current state of the object. However, developers nowadays require not only access to current valid data, but they also require having overview of properties, structures and values at any time point or time interval. Management of backups and log files is completely inappropriate, complicated and requires too much processing time. Temporal database brings possibilities and opportunities by adding time attributes limiting the validity and even transaction information. The aim is to store, monitor and evaluate information about all states of the objects during the life-cycle. Our system can store the information after the delete operation, if necessary.

Standard temporal database support is based on the object level; one row represents the whole state at a defined time point or time interval. However, our system is based on the column attribute level, the whole state is created by the grouping of the properties and states of the attributes. Critical factor for the new development is the processing time to get required reliable data. This paper deals with the principles and characteristics and describes implementation methods to provide the complex temporal data management. The developed system is compared with the existing structures based on the performance, the time to get required data as well as the size of the database and provides very good performance results.

Proposed methods and structure can be extended by the system processing transaction time.

The temporal data are usually large; the processing requires sophisticated access methods. In the future development, we will focus on the index structures creation, which should improve the performance of the model, too.

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.

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

This work was partially supported by the project: ***Creating a new diagnostic algorithm for selected cancers***, ITMS project code: 26220220022 co-financed by the EU and the European Regional Development Fund.

The work is also supported by the project VEGA 1/1116/11 - ***Adaptive data distribution***.

References

[1] MATIASKO, K., VAJSOVA, M., ZABOVSKY, M., CHOCHLIK, M.: *Database systems*. EDIS, 2008. ISBN: 9788080708207

[2] DATE, C.: *Date on Database*. Apress, 2006. ISBN: 9781590597460

[3] JOHNSON, T., WEIS, R.: *Managing Time in Relational Databases.* Morgan Kaufmann, 2010. ISBN: 9780123750419

[4] SNODGRASS, R.: *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann Publishers, San Francisco, 2000.

[5] KVET, M., LIESKOVSKY, A., MATIASKO, K.: *Uni-temporal and Bi-temporal Table*. IEEE Conference Digital Technologies 2013, Zilina, ISBN: 9781479909223.

[6] SNODGRASS, R.: *Developing Time-Oriented Database Applications in SQL*, Morgan Kaufmann, 1999. ISBN: 1558604367.

[7] KVET, M., MATIASKO, K.: *Temporal Data Management.* Conference IARIA ICCGI 2013, Nice, ISBN: 9781612082837

[8] KVET, M., LIESKOVSKY, A., MATIASKO, K.: *Temporal Data Modelling.* IEEE Conference ICCSE 2013, Colombo, 9781467344623

[9] KVET, M., MATIASKO, K.: *Conventional and Temporal Table – Temporal Table Model*. Conference ARSA, Zilina, 2012. ISBN: 9788055406060

[10] KVET, M., MATIASKO, K.: *Temporal Data Modeling – Future Valid Data Processing*. Conference DICTAP 2013, Ostrava, ISBN: 978989130609

[11] KVET, M., MATIASKO, K.: Management of Temporal System – Column Level. *Intern. J. of New Computer Architectures and their Applications*, vol. 3, No. 3, 2013. ISSN: 22209085.