Temporal Data Management

Uni-temporal Table Modelling Update

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

Anton Lieskovský
Department of Informatics
Faculty of Management Science, University of Zilina
Zilina, Slovakia
Anton.Lieskovsky@fri.uniza.sk

***Abstract*— Today, it is necessary to store not only actual data – data that are valid at this moment, but also historical data, by which the progress and frequency of changes can be monitored. Managing historical data offers creating future prognoses and analyses. Temporal tables (mostly modelled using uni-temporal and bi-temporal tables) in comparison with conventional tables can process and retain the information of the validity in the past. Using procedures, functions, triggers, cursors – snapshot of the database object or whole database at any time point can be reconstructed easily. Thus, each database object is not represented by only one row, but by the set of rows representing the same object during a different period of time. This paper deals with the principles of temporal data modelling and offers the solution based on uni-temporal table model. It contains the implementation methods based on changes monitoring and deals with the problem of managing undefined states of the objects. Operations implemented in temporal system is compared with the conventional model.**

Keywords-conventional table; temporal table; uni-temporal model; valid time;

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.

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.

Most of the data in the database represent actual states. However, properties of objects and states are changed over time - customer changes its status, address, products are modified and updated. If the object state is going to be changed, data in the database are updated and the database will still contain only currently applicable object states. But everything has time evolution, thus, history and future that can be useful to store. History management is very important in systems processing very important or 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 make future decisions.

Historical data are now possible to be saved using log files and archives. Thus, the historical data can be obtained, but it is complicated process. In addition, management requires quick and reliable access to data defined by any time point, but also getting information about the attributes changes in the future without significant time delays [10] [11].

Temporal data processing is not a new problem. Early in the development of the databases transaction, log files were created and database was regularly backed up. Log files were usually deleted after the backup, because all transactions were recorded in the production database. Thus, it was possible to obtain an image database at any time point. However, these data were in the raw form (raw material) and handling them was difficult, lasted too much time. 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.

Fundamental paradigm of database systems used since the beginning of the data processing focuses on the actual data processing [10] [11] [12].

The situation in the computer field changed significantly in the early 80-ies of the 20th century- price of the disk storage space decreased allowing greater and easier way to save backups. So, there was an opportunity to compare multiple images (backup) from different time periods.

Each backup is a snapshot of the database table or the whole database. However, if the values do not change (or it is not necessary to store historical values of them), too much duplicities are stored. Later, the first concept of the data warehouse based on the database table level was created by Barry Devlin and Paul Murphy [6].

Recent history shows the potential opportunities of historical data processing, which could be faster and more efficient than managing backups or log files. The main disadvantage of the above-mentioned ways is the need of the administrator 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 backup to get the database snapshot in 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 was above the acceptable level. An important task for administrators was to define the time frame between two backup of the database. If the interval is too large (assuming backups without using their own log files - they are too large for more images), not all operations are stored there, e.g. the insert and also delete of the record could be between two images, so user has no information about the existence of the object in the database. Another example is multiple updates of the same record; see Figure 1. The opposite way, if the interval was too small, large images containing a lot of the same attribute values were created, which include high demands on disc storage. Interval between two backups can also be defined dynamically, but the problem with uncaught of some changes remains unresolved. Another solution is to delete old images, which is unacceptable because of the management requirements [8].

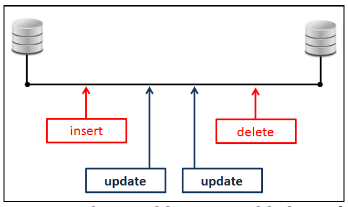


Figure 1. Backup problem - possible loss of data

It is not enough to find a faster solution for historical data processing. What does the term “faster” mean? To order shorten the processing time from days to hours, from tens of hours to a couple of hours? The usage of that solution is unacceptable and inapplicable. The aim is to create support for the temporal data, that the difference between the processing in the currently valid data and historical data is minimized. Thus, it is necessary to create a system with easy data. However, historical data management is not a complete temporal system, because it must be allowed to process values, the validity of which begins in the future. If the begin time of the validity of the object occurred, the database system must be able to update the data without user intervention.

System requirements can be divided into two parts with special aspects [6] [13] [14]:

1. aspect of usability (easy methods) – the aim is to provide access to outdated information as easily and quickly as to the actual values. Transactions for managing temporal data must be as simple as for current data processing. Moreover, it is necessary to define a way to combine the past and present.

2. aspect of performance (speed and correctness of results) requires results in the same form as when accessing the actual data with adequate processing time. The difference in accessing the object at any point in time should be minimal.

Following the history of computer systems for data management, we come to the conclusion that the databases have been developed and deployed for the current values administration - currently valid data processing. None of the above-mentioned solution does not have the structure to represent objects and their states during the time, support for managing attribute value changes in the temporal dimension. However, developers require those functionality and data access.

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 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 processed period. If the valid time of the object is defined by time interval, the transaction must include time period - 2 time point values - begin and end timestamps (dates). 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 [2] [6] [8].

This paper consists of five sections. The second deals with the structure of conventional and temporal table, the third describes the implementation techniques and methods. The section named “Experiments” deals with the improvements of the model and the current performance. The last one is the conclusion.

II. CONVENTIONAL AND TEMPORAL TABLE

Row in a relational database table can be defined in three different ways using time (Figure 2). "ID" is a unique identifier, "PK" refers to a primary key. "BD1" and "ED1" are a pair of columns defining the beginning and end value of the period, "BD2" and "ED2" define the second time interval. The primary key can be defined composite without changing the functionality of the methods used. The first model does not use time for definition. This is a standard model used today, called the conventional model. The primary key is defined by the attribute "ID". All not key attributes in the table, regardless of their number, are merged into a common block called "data" [1] [3] [4] [5].

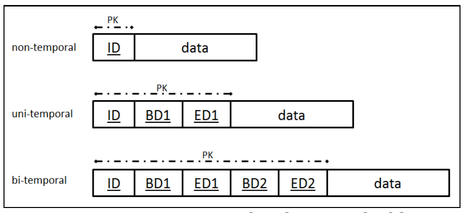


Figure 2. Conventional and temporal table

In this paper, we use the standards used in scientific articles, the names of the attributes forming the primary key are underlined. The following table shows the object identified by an attribute value ID = 1 in the time period <September 2012 - December 2013>. We can see that the attribute "data" has the value “123” at the time <September

2012 - June 2013>, in the second part of the interval, attribute "data" has the value of the “234”. The representation of the time interval is the closed-closed type.

TABLE I. UNI-TEMPORAL

ID	BD1	ED1	data
1	**September 2012**	**June 2013**	**123**
2	January 2013	November 2014	555
1	**July 2013**	**December 2013**	**234**

The problem occurs, if the end date of the time period is unknown. There are two opportunities, in general. The first is to deal with undefined values - NULL. In this case, if there is a request to obtain the current state of the object, respectively overall database, we cannot use standard methods to manipulate time and we have to add more conditions. The second solution is to replace an undefined time with the maximal value that can be used - "December 9999", or other variants depending on time granularity. If we need to update the state of the object, the attribute value defining the end of the validity (attribute "ED") is replaced by the actual time value.

The last - third basic model – bi-temporal model - is based on the concept of uni-temporal tables, but uses two time intervals. Thus, it allows not only defining several lines for one object, but also multiple rows for a particular object's state at the time. The reason for this model is the need to record an updated status in the past.

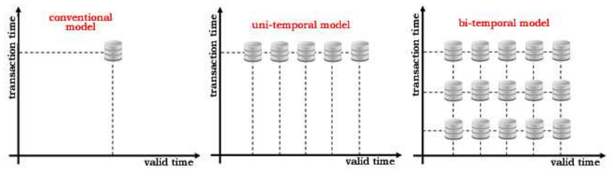
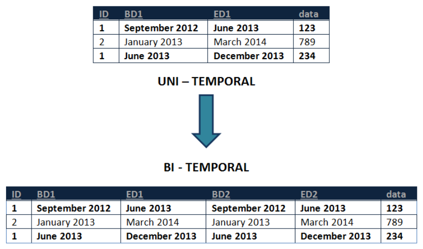


Figure 3. Time representation

The method for transformation uni-temporal model to bi-temporal is based on the same principle as the conversion of the conventional model (conventional model) to uni-temporal model - two additional attributes defining a timestamp (begin and end time of the validity) are added to the primary key. The values of the new attributes after the transformation can be identical to the first time interval defined by attributes “BD1” and “ED1” (Figure 4).

The transformation from bi-temporal model to uni-temporal model is also possible, but there is possibility of the loss of the updated data [6] [7] [8].



ID	BD1	ED1	data
1	**September 2012**	**June 2013**	**123**
2	January 2013	March 2014	789
1	**June 2013**	**December 2013**	**234**

UNI – TEMPORAL

BI - TEMPORAL

ID	BD1	ED1	BD2	ED2	data
1	**September 2012**	**June 2013**	**September 2012**	**June 2013**	**123**
2	January 2013	March 2014	January 2013	March 2014	789
1	**June 2013**	**December 2013**	**June 2013**	**December 2013**	**234**

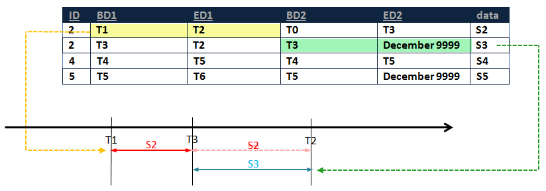
Figure 4. Transformation of the uni-temporal table to bi-temporal table

Figure 4 shows the principle of bi-temporal modelling. Primary key of this model consists of three logical units:

• **The object identifier (ID).**

• **Interval (BD1, ED1)** - the time during which the object has been describing the characteristics of the row, e.g. the period during which a customer has the characteristics - name, address, status, etc.

• The last component of the logical primary key is a **pair of dates** (or timestamps according to the representation of a time granularity of data). These dates limit the period during which we believe the value of the row is correct. This component limits the time interval defined by the second component (BD1, ED1).



ID	BD1	ED1	BD2	ED2	data
2	T1	T2	T0	T3	S2
2	T3	T2	T3	December 9999	S3
4	T4	T5	T4	T5	S4
5	T5	T6	T5	December 9999	S5

Figure 5. Bi-temporal table

Figure 5 shows the principle of bi-temporal table modelling.

III. UNI-TEMPORAL TABLE IMPLEMENTATION

Transformation of the conventional table to temporal model is not trivial problem and needs special structures and resources. In addition, the requirement of the users is to provide compatibility, easy manipulation and proper time consumption. Therefore, the triggers, procedures and functions must be declared, the original tables can be transformed to views (if necessary). Each database record is defined by the primary key. In most cases, it is the unique identifier (ID), sometimes, we use composite primary key. Conventional database tables transformation recommends single attribute primary key in every table to create a common temporal table for all tables containing temporal attributes. Thus, ID is suitable; each record can be clearly defined and referenced. Moreover, the ID does not have special denotation and the need for its change is irrelevant, e.g. personal identification number contains the birthdate and if the mistake in time of insert occurs, the record must be updated. Thus, the best way is to create sequence for ID; trigger before insert sets the correct value.

Information of any change of temporal column is recorded in the table managing changes - temporal_table. However, it contains information only about table containing temporal column. This table consists of these attributes (see also Figure 5) [7]:

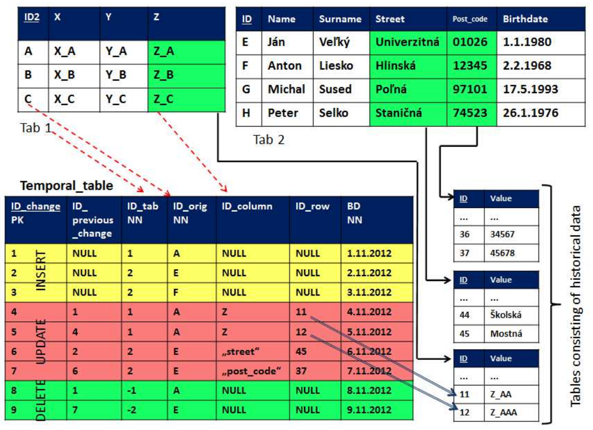
- **ID change**
- **ID previous change** – references the last change of an object identified by ID. This attribute can also have NULL value that means, the data have not been updated yet, so the data were inserted for the first time in past and are still actual.

- **ID_tab** – references the table, record of which has been processed by DML (Data modelling language) 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** – validity of the new state of an object starting.

```
CREATE OR REPLACE TRIGGER Upd_tab_2_trigger
BEFORE UPDATE ON TAB2
FOR EACH ROW
BEGIN
 if (:old.street<>:new.street)  then
    --data processing
end if;
if (:old.post_code<>:new.post_code)  then
    --data processing
end if;
END;
/
```

Figure 6. Update trigger

The following figure shows the data representation and manipulation.



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

ID	Name	Surname	Street	Post_code	Birthdate
E	Ján	Veľký	Univerzitná	01026	1.1.1980
F	Anton	Liesko	Hlinská	12345	2.2.1968
G	Michal	Sused	Poľná	97101	17.5.1993
H	Peter	Selko	Staničná	74523	26.1.1976

ID_change PK	ID_ previous _change	ID_tab NN	ID_orig NN	ID_column	ID_row	BD NN
1	NULL	1	A	NULL	NULL	1.11.2012
2	NULL	2	E	NULL	NULL	2.11.2012
3	NULL	2	F	NULL	NULL	3.11.2012
4	1	1	A	Z	11	4.11.2012
5	4	1	A	Z	12	5.11.2012
6	2	2	E	„street“	45	6.11.2012
7	6	2	E	„post_code“	37	7.11.2012
8	1	-1	A	NULL	NULL	8.11.2012
9	7	-2	E	NULL	NULL	9.11.2012

ID	Value
...	...
36	34567
37	45678

ID	Value
...	...
44	Školská
45	Mostná

ID	Value
...	...
11	Z_AA
12	Z_AAA

Figure 7. Temporal_table model [7]

A. *Insert trigger*

New record containing information about the change of the temporal column is inserted into the temporal table after inserting into conventional table. These operations are provided by insert trigger. The new value for attribute “ID_change” is set using the sequence. Value of “ID_previous_change” attribute is null, which means, the new data have been inserted (for the first time). There is no reference to old value of the attributes, so the “ID_row” and “ID_column” also contain null value (example for TAB1) (see also Figure 5) [7]:

```
INSERT INTO temporal_table (ID_change, ID_previous_change,
                 ID_tab, ID_orig, ID_column, ID_row, BD)
  values (ID_TEMPORAL_TABLE_sequence.nextval, NULL, 1,
           ID_tab1_sequence.currval, NULL, NULL, sysdate);
```

Figure 8. Temporal table insert

B. *Update trigger*

Updating existing data requires saving old data – not the whole row, but only changed temporal attribute values. The original table consists of the actual data, so the data manipulation – actual snapshot is easy to get. Historical data – the snapshot of the whole database, database table or only object – must be also accessible, but are obtained by passing historical conditions defined by insert, delete or update statement. Thus, the update trigger is started before update. First of all, the data that will be changed are stored in the table consisting only of the ID of the record and the value itself [7].

```
INSERT INTO TAB_Z_COLUMN (ID, Z) values
                (ID_TAB_Z_COLUMN_sequence.nextval, :old.Z);
```

Figure 9. Updating column "z"

Then, the reference to the change is stored in the temporal table (Figure 5) [7]:

- **ID_change** is set using the sequence and trigger.
- **ID_previous_change** is maximum of “ID_change “used for those ID original and ID table *(select max(ID_change) INTO cislo from temporal_table where ID_orig=:old.ID AND ID_tab=1;).*
- **ID_column** references the temporal column, data of which is going to be changed.
- **ID_row** associates the table with historical values.

C. *Delete trigger*

The task of the trigger starting before delete is to save old data to the table for deleted objects. The information about delete is also inserted to the temporal table; “ID_tab” now has the negative value. The relevance of it will be described later [7].

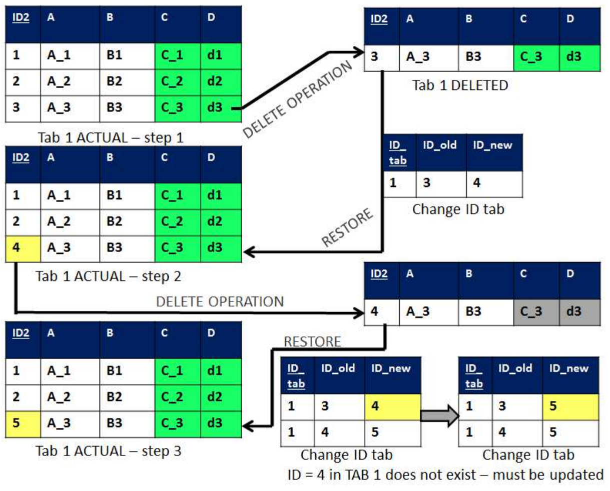
```
INSERT INTO TAB1_DELETED(ID, X, Y, Z) values(:old.ID, : old.X, :
old.Y, : old.Z);

INSERT INTO temporal_table (ID_change, ID_previous_change, ID_tab,
                            ID_orig, ID_column, ID_row, BD)
      values (ID_TEMPORAL_TABLE_sequence.nextval, max_change ,
          -1, : old.ID, NULL, NULL , sysdate);
```

Figure 10. Body of the trigger - delete

D. *RestoreData*

Deleted data recovery is a further problem. If the object data in the main table are not valid or we do not know the correct value in this moment, the object must be deleted, respectively relocated to table of deleted data. However, if the object attributes are again valid, we need to restore data to the original table. The problem causes a trigger which sets the new value of ID before inserting, so the old value of ID cannot be used. However, the database structure should have information, that the ID has been changed, but the object is the same. In addition, if the user is still using the old identifier, the methods working with original (old) ID would not work correctly. There can be used more solutions, one of them is to disable temporary the trigger for ID, restore the data and enable it. This solution is absolutely incorrect.



Tab 1 ACTUAL – step 1

ID2	A	B	C	D
1	A_1	B1	C_1	d1
2	A_2	B2	C_2	d2
3	A_3	B3	C_3	d3

Tab 1 DELETED

ID2	A	B	C	D
3	A_3	B3	C_3	d3

Change ID tab

ID_tab	ID_old	ID_new
1	3	4

Tab 1 ACTUAL – step 2

ID2	A	B	C	D
1	A_1	B1	C_1	d1
2	A_2	B2	C_2	d2
4	A_3	B3	C_3	d3

ID2	A	B	C	D
4	A_3	B3	C_3	d3

Tab 1 ACTUAL – step 3

ID2	A	B	C	D
1	A_1	B1	C_1	d1
2	A_2	B2	C_2	d2
5	A_3	B3	C_3	d3

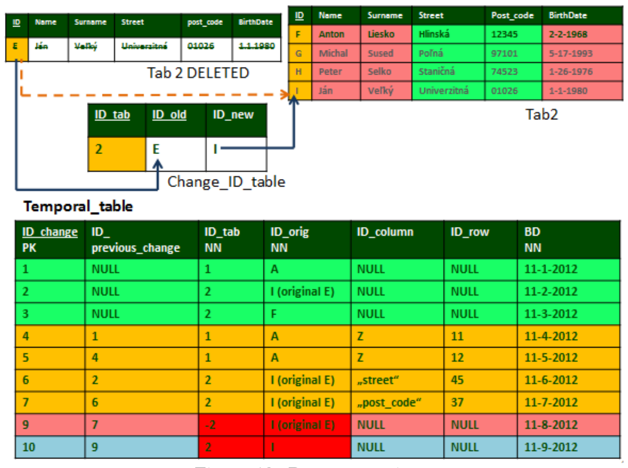
Change ID tab

ID_tab	ID_old	ID_new
1	3	4
1	4	5

Change ID tab

ID_tab	ID_old	ID_new
1	3	5
1	4	5

Figure 11. Principle of data restoration [7]



Tab 2 DELETED

ID	Name	Surname	Street	post_code	BirthDate
E	~~Ján~~	~~Veľký~~	~~Univerzitná~~	~~01026~~	~~1.1.1980~~

Tab2

ID	Name	Surname	Street	Post_code	BirthDate
F	Anton	Liesko	Hlinská	12345	2-2-1968
G	Michal	Sused	Poľná	97101	5-17-1993
H	Peter	Selko	Staničná	74523	1-26-1976
I	Ján	Veľký	Univerzitná	01026	1-1-1980

Change_ID_table

ID_tab	ID_old	ID_new
2	E	I

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	I (original 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	I (original E)	„street“	45	11-6-2012
7	6	2	I (original E)	„post_code“	37	11-7-2012
9	7	-2	I (original E)	NULL	NULL	11-8-2012
10	9	2	I	NULL	NULL	11-9-2012

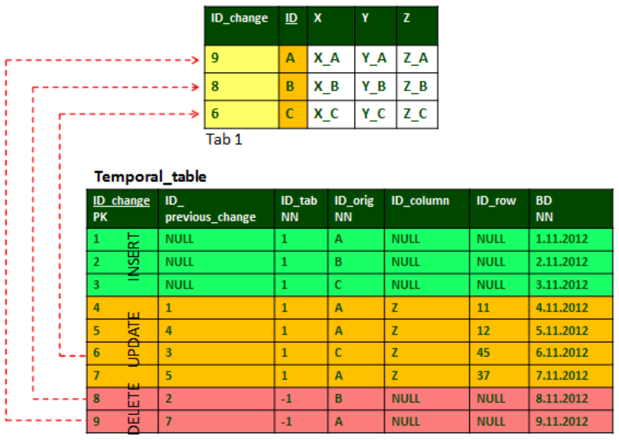
Figure 12. Data restoration

Our solution uses the new ID (Figure 11, Figure 12), old and new object is connected together. Thus, these steps must be done [7]:

- Data relocation from „deleted objects table“ to main table (insert + delete).
- Insert info about data restoration into temporal table.
- Update old „ID_orig“ (temporal table) to actual.
- Insert into „change_ID_table“ new values – „ID_tab“, „ID_old“, „ID_new“.
- Update „the change_ID_table“ - if necessary. The point is that the data could be relocated several times and therefore the old reference (but stored in the attribute „ID_new“) is not actual. Figure 12 describes the problem.

IV. EXPERIMENTS

The overall adjustment and optimization of the procedures, functions, triggers and the model itself, we got very good values of the processing time; the worst result (update and delete statement) is a slowdown of 34% in comparison with conventional table model. It should be noted, however, that the traditional model does not retain any information about changes of the database. Our designed solution stores also old (historical) values - any snapshot of the database (database object) defined by time point can be got easily using programmed methods.

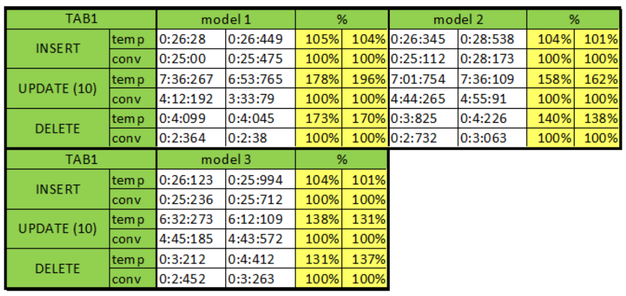


ID_change	ID	X	Y	Z
9	A	X_A	Y_A	Z_A
8	B	X_B	Y_B	Z_B
6	C	X_C	Y_C	Z_C

ID_change PK	ID_ previous_change	ID_tab NN	ID_orig NN	ID_column	ID_row	BD NN
1	NULL	1	A	NULL	NULL	1.11.2012
2	NULL	1	B	NULL	NULL	2.11.2012
3	NULL	1	C	NULL	NULL	3.11.2012
4	1	1	A	Z	11	4.11.2012
5	4	1	A	Z	12	5.11.2012
6	3	1	C	Z	45	6.11.2012
7	5	1	A	Z	37	7.11.2012
8	2	-1	B	NULL	NULL	8.11.2012
9	7	-1	A	NULL	NULL	9.11.2012

Figure 13. Principle of model 3

Figure 14 shows the total execution time of DML operations (insert, delete and update) for conventional and temporal tables. The total number of operations is 10 000. The experiments were provided using Oracle 11g instance.



TAB1		model 1		%		model 2		%	
INSERT	temp	0:26:28	0:26:449	105%	104%	0:26:345	0:28:538	104%	101%
	conv	0:25:00	0:25:475	100%	100%	0:25:112	0:28:173	100%	100%
UPDATE (10)	temp	7:36:267	6:53:765	178%	196%	7:01:754	7:36:109	158%	162%
	conv	4:12:192	3:33:79	100%	100%	4:44:265	4:55:91	100%	100%
DELETE	temp	0:4:099	0:4:045	173%	170%	0:3:825	0:4:226	140%	138%
	conv	0:2:364	0:2:38	100%	100%	0:2:732	0:3:063	100%	100%

TAB1		model 3		%	
INSERT	temp	0:26:123	0:25:994	104%	101%
	conv	0:25:236	0:25:712	100%	100%
UPDATE (10)	temp	6:32:273	6:12:109	138%	131%
	conv	4:45:185	4:43:572	100%	100%
DELETE	temp	0:3:212	0:4:412	131%	137%
	conv	0:2:452	0:3:263	100%	100%

Figure 14. Research results

Model 1 does not use the table with maximal values of changes, data are inserted into historical tables only if they have not been there yet. Model 2 uses table with maximal values of changes and historical data are always saved.

Model 3 stores the value of the identifier of the last change of the object directly in the main production table. This is due to a reduction in the number of records. Specifically, the table “Change_ID_tab” stores values for all temporal objects, and also contains data about deleted objects. If the number of deleted objects is great, the reduction of processing time is significant.

The overall slowdown of this model is:

- insert operations – 3%,
- delete operations – 34%,
- update operations – 34%.

As it was already mentioned, the results of the experiments can be considered satisfactory, because of the storing the entire history of temporal columns.

V. Conclusion

Each instance in the conventional database is represented by one row. Temporal database concept offers new opportunities by adding additional time attributes limiting the validity of the object.

Temporal support brings new possibilities by extending the primary key with time interval. Two different time period defining the same object validity cannot be valid at the same time – cannot overlap. The aim of the temporal databases is to store information about all states of the objects during their life cycle, even after DELETE operation.

Comparison of the processing time of the conventional methods and temporal model is the root for the further research. Experiments, that have been made, show that the operation UPDATE is the critical factor. If we update the row, we need to save old value (insert it into historical table).

Proposed methods can be extended by the special attribute defining the end of the validity. This will create possibility of defining period during which the state of the object is not valid without the necessity of deleting it.

The begin time of the validity can be also in the future. In the future development, we are going to create methodology for future states management. Data valid in the future will be stored in special tables. New record will be automatically inserted into the database at the time of the beginning of the validity.

VI. Acknowledgment

This contribution 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 work was 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 supported by the project VEGA 1/1116/11, Adaptive data distribution.

References

[1] C. J. Date, “Date on Database”. Apress, 2006.

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

[3] Ch. S. Jensen, “Introduction to Temporal Database Research”

[4] Ch. S. Jensen and R. T. Snodgrass, “Temporally Enhanced Database Design”

[5] P. N. Hubler and N. Edelweiss, “Implementing a Temporal Database on Top of a Conventional Database”

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

[7] M. Kvet, A. Lieskovský, and K. Matiaško, “Temporal data modelling”, 2013.(IEEE conference ICCSE 2013, 4.26. – 4.28.2013), pp. 452 - 459

[8] M. Kvet and K. Matiaško, “Conventional and temporal table”, EDIS, 2012. (virtual conference ARSA, 12.3. – 12.7.2012), pp. 1948-1952

[9] N. Mahmood, K. Rizwan, and S. A. K. Bari, “Fuzzy-Temporal Database Ontology and Relational Database Model”, 2012.

[10] K. Matiaško, M. Vajsová, M. Zábovský, and M. Chochlík, “Database systems”, EDIS, 2008.

[11] K. Matiaško, M. Vajsová, M. Zábovský and M. Chochlík, “Database systems and technologies”, STU, 2009.

[12] G. Ozsoyoilu and R. T. Snodgrass, “Temporal and Real-Time Databases: A Survey”, 1995.

[13] R. T. Snodgrass, “Developing Time- Oriented Database Applications in SQL”, Morgan Kaufmann, 1999.

[14] R. Snodgrass, “Adding Valid Time to SQL/Temporal”