ABSTRACT

Database systems used nowadays must allow not only current valid data processing, but also managing historical data, respectively data, which will be valid in the future. Moreover, it must be able to provide automated change of the values of the attributes. At present, database systems defined for managing temporal data are based on the principle of a new image of the object creation. Some attribute values, however, do not change their values over the time or are updated very rarely, and therefore it is not necessary to record the new values for these attributes. This paper deals with the principle of temporal data modelling based on column level, not the whole object. It describes the principles, required methods, procedures, functions and triggers to provide the functionality of this system. It also defines the possible implementations and offers the solution to get the snapshot of the database or the object whenever during the existence.

KEYWORDS

Conventional table; Fully temporal table; Future valid data processing; Jobs;

1 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.

2 HISTORY AND EASE OF USE

The need to manage and model temporal data has come with the first database systems. Developers realized that it is necessary to store the historical data to provide possibility to restore the database if the database integrity is not correct, the medium is damaged and so on. Backups and log files were considered the main elements of historical data in the past. However, most programmers used to ignore them, because when using them – it indicated severe problem and the need to restore last correct version of the database. Access to historical data using those methods - the image of the object at a particular time - was complicated, lasted too much time, required the administrator intervention and the values were necessary to be reconstructed. Moreover, if the backup was created, the earlier log files were usually deleted. Thus, the data could be provided only in the time when the database backup was created.

Later, data storage and access technology made possible to manage massively larger volumes of data than ever before, much of that historical data were stored on-line and organized it in two different ways. Backups were stacked on top of
one another and turned into data warehouses. On the other hand, log files were supplemented with foreign keys and turned into data marts. But warehouses and data marts are macro structures. They are structures of temporal data at the level of databases and their instances.

However, these data were in the raw form (raw material) and handling them was difficult, lasted too much time.

Fundamental paradigm of database systems used since the beginning of the data processing focuses on the actual data processing. [1] [2] [3] [4] [6]

The situation in the computer field changed significantly in the early 80-ies of the 20th century - price of the disc 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. [2]

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 backups was above the acceptable level. An important task for administrators was to define the time frame between two backups 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 update of the same record – fig.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, what is unacceptable because of the management requirements. [7] [11]

As we will see in the following sections, this solution brings a lot of other disadvantages which are associated with several problems.

![Figure 1. Backup problem - possible loss of data](image)

### 3 TEMPORAL DATABASE MODELLING

Current concept of temporal model is based on an extension of time table attributes that define the validity period. Thus, the primary key contains only the identifier, but also information about the period during which the record has a given characteristic. However, do not to forget the fact that the completed primary key is referenced as a foreign key. It can cause problems with the time period updating (fig. 2).
Moreover, object attributes defining the validity does not need to affect relationships and thus referencing the time attribute cannot be necessary (even useless).

4 STRUCTURE OF TEMPORAL TABLE

Temporal extension of the standard conventional (non-temporal) model can be in principle created in two ways - by defining validity and transaction time. Temporal systems can thus be divided into two basic types:

Uni-temporal system is based on the time of the validity. Row in the database table thus defines the object and also the time which may be defined by an interval (start and end time of the interval), or only by the beginning. In that case each new record determines the end of the previous record based on the same object. This way - just the begin time of the period - is used in our developed column-level solution. In this case, however, developer should not forget to manage and report states, during which the object is undefined.

Next table shows the example of the data using the uni-temporal model.

Table 1. Uni-temporal table

<table>
<thead>
<tr>
<th>ID</th>
<th>BD</th>
<th>ED</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Jan 2012</td>
<td>Mar 2012</td>
<td>111</td>
</tr>
<tr>
<td>3</td>
<td>Feb 2012</td>
<td>Nov 2012</td>
<td>123</td>
</tr>
<tr>
<td>4</td>
<td>Feb 2012</td>
<td>Feb 2013</td>
<td>764</td>
</tr>
<tr>
<td>2</td>
<td>Mar 2012</td>
<td>Dec 2012</td>
<td>222</td>
</tr>
<tr>
<td>3</td>
<td>Nov 2012</td>
<td>Feb 2013</td>
<td>890</td>
</tr>
</tbody>
</table>

Figure 3 shows the structure of uni-temporal model, where ID is the identifier of the object (for simplicity, the definition of the object is not a composite, but if there is an necessity to create it compositely, it would not affect the principles).

Thus, primary key contains one (BD1) or two (BD1, ED1) attributes defining the temporal validity. Non-key attributes are grouped together with the common name data. If we want to select the values (characteristics), we should define not only the object, but also the time point or the time interval of the validity. It should be mentioned that the result of the processing can contain more rows for the specific object, it always depends on the number of changes over the corresponding object during the time defined by the SELECT statement – For example, changing addresses, price trends and so on.

The second temporal system is bi-temporal model. This concept extends the previous model by adding the transaction time in the system (BD2, ED2).

Table 2. Bi-temporal table

<table>
<thead>
<tr>
<th>ID</th>
<th>BD1</th>
<th>ED1</th>
<th>BD2</th>
<th>ED2</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>May 2012</td>
<td>Aug 2012</td>
<td>May 2012</td>
<td>Aug 2012</td>
<td>123</td>
</tr>
<tr>
<td>2</td>
<td>Jul 2012</td>
<td>Nov 2012</td>
<td>Jul 2012</td>
<td>Dec 9999</td>
<td>345</td>
</tr>
<tr>
<td>1</td>
<td>Aug 2012</td>
<td>Jan 2013</td>
<td>Nov 2012</td>
<td>Dec 9999</td>
<td>457</td>
</tr>
</tbody>
</table>

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

- The object identifier (ID).
- Interval \((BD_1, ED_1)\) - 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 \((BD_1, ED_1)\).

Both systems are fully temporal, allowing not to work only with history, but also with data valid in the future. But the important thing is to ensure that individual objects are defined by only one row in each time point.

The area of temporal databases focuses mainly on these three interval relations [6] [15]:

1. Relationship [intersects] is important for the transactions, that add new records to the database. Time interval defining the validity of the record must be disjoint with all already defined intervals of a given object. There cannot be valid two or more versions (episodes) at the same time. Update and delete transactions again use this method to find the record, the validity of which contains a user specified time.
2. Relationship [before] is used to distinguish and sort the individual episodes.
3. The basis for versions comparison and unification is the relationship [meets].

Transformation of the conventional data model to a temporal is not easy, although it can seem to be. The main problem is changing the unsteadiness of timed attributes (e.g. the address is changed more often than surname of woman), which generates a lot of value duplicity, if the data model is not designed well. The easiest way is to add begin date or the couple - begin and end date attribute. Usually, only one value of attribute is updated in the time of change. (Table 3)

<table>
<thead>
<tr>
<th>ID</th>
<th>BD</th>
<th>ED</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>An</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>May 2011</td>
<td>Dec 2011</td>
<td>X</td>
<td>F</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>1</td>
<td>Jan 2012</td>
<td>Feb 2012</td>
<td>X</td>
<td>F</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>1</td>
<td>Feb 2012</td>
<td>Apr 2012</td>
<td>X</td>
<td>L</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>1</td>
<td>Apr 2012</td>
<td>Sep 2012</td>
<td>X</td>
<td>L</td>
<td>B</td>
<td>E</td>
</tr>
</tbody>
</table>

5 UNI-TEMPORAL MODEL BASED ON THE BEGIN TIME OF THE VALIDITY

A special model of uni-temporal system is a solution that contains only one time attribute that is part of the primary key. This means that any change of the corresponding object determines the validity of the prior state. The following figure
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 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 (more about the time interval modeling can be found in [9]).

![Figure 6. Types of uni-temporal table modelling](image)

<table>
<thead>
<tr>
<th>ID</th>
<th>BD</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sep 2013</td>
<td>data</td>
</tr>
<tr>
<td>1</td>
<td>Dec 2013</td>
<td>data</td>
</tr>
</tbody>
</table>

Another solution is based on the using NULL values to show the attribute is undefined. However, it can be only used, if the value NULL does not have the special denotation (fig.8).

The figure 8 also shows, that between the September and December 2013, the value of the attribute data_2 is undefined.

![Figure 7. Data model processing also invalid data](image)

<table>
<thead>
<tr>
<th>ID</th>
<th>BD</th>
<th>valid</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sep 2013</td>
<td>Y</td>
<td>data</td>
</tr>
<tr>
<td>1</td>
<td>Dec 2013</td>
<td>N</td>
<td>data</td>
</tr>
</tbody>
</table>

![Figure 8. Unknown data values of the attribute processing](image)

<table>
<thead>
<tr>
<th>ID</th>
<th>BD</th>
<th>data_1</th>
<th>data_2</th>
<th>data_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sep 2013</td>
<td>value</td>
<td>NULL</td>
<td>value</td>
</tr>
<tr>
<td>1</td>
<td>Dec 2013</td>
<td>value</td>
<td>value</td>
<td>NULL</td>
</tr>
</tbody>
</table>

6 IMPORTANCE OF NOTATION 9999

The notation 9999 or its modifications based on the granularity of the data, represents the latest date that can be used in current database systems. It can be used only by end date of the period, the meaning is “later than now”. It means that we do not know the exact time of the end validity, but important fact is that this moment has not come yet – it is greater than Now().

This is also one of the reasons, why not to use the NULL values. On one hand, this value does not reflect the future and, secondly, we cannot use the power of the functions for comparing time. Simply, NULL value should be used for unknown data, whereas this information is partially known. So the implementation is a real data. We assume, that he date 12/31/9999 does not represent far-off New Year’s Eve, but it is used for a special purpose. In the case of the validity, we usually insert data without knowing when the time of the new version may end. For example, when the product is added to the database, we usually do not know when the price or other characteristics will be changed. We can only assume that this change will occur, but surely in the future. [2]
7 TEMPORAL MODEL – COLUMN LEVEL

Solution for temporal management should be universal, not only in terms of usability in practice, but also in terms of 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 system itself thus contains special structure for temporal storage of data, the external level works with non-temporal data. The structure of the main tables is almost the same with the conventional model.

The following figure 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. Column, which changes need to be monitored, is temporal. If the value is changed, information about the update is stored in the temporal table and historical value is inserted into the table containing historical values. Each temporal column has the own historical table.

- **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 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 validity.

The following figure shows the data representation and manipulation.

![Figure 9. Temporal system](image)

Temporal table consists of the next mentioned attributes [9] [10] – see also fig. 10:

- **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 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 validity.

The following figure shows the data representation and manipulation [9]

![Figure 10. Temporal model representation and manipulation](image)

The next chapters describe the principles of the most important operations providing the system.

7.1 Insert

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. 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 \( ID_{\text{row}} \) and \( ID_{\text{column}} \) also contain null value (example for TAB1) (see also fig. 10)

### 7.2 Update

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 are going to be changed, are stored in the table consisting only of the \( ID \) of the record and the value itself.

Then, the reference to the change is stored in the temporal table (see also fig. 10) [9]:

- \( ID_{\text{change}} \) is set using the sequence and trigger.
- \( ID_{\text{previous\_change}} \) is maximum of \( ID_{\text{change}} \) used for those \( ID_{\text{original}} \) and \( ID_{\text{table}} \) (select max\( ID_{\text{change}} \) from temporal_table where \( ID_{\text{orig}}=\text{old.ID} \) AND \( ID_{\text{tab}}=1 \)).
- \( ID_{\text{column}} \) references the temporal column, data of which is going to be changed.
- \( ID_{\text{row}} \) associates the table with historical values.

<table>
<thead>
<tr>
<th>( ID_{\text{change}} )</th>
<th>( ID_{\text{previous_change}} )</th>
<th>( ID_{\text{tab}} )</th>
<th>( ID_{\text{orig}} )</th>
<th>( ID_{\text{column}} )</th>
<th>( BD )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NULL</td>
<td>1</td>
<td>A</td>
<td>NULL</td>
<td>11-1-2012</td>
</tr>
<tr>
<td>2</td>
<td>NULL</td>
<td>2</td>
<td>E</td>
<td>NULL</td>
<td>11-2-2012</td>
</tr>
<tr>
<td>3</td>
<td>NULL</td>
<td>2</td>
<td>F</td>
<td>NULL</td>
<td>11-3-2012</td>
</tr>
</tbody>
</table>

**Figure 11.** Example of the insert into temporal table after the operation insert

\[
\text{INSERT INTO temporal\_table} \\
\quad (ID_{\text{change}}, ID_{\text{previous\_change}}, \ID_{\text{tab}}, ID_{\text{orig}}, ID_{\text{column}}, BD) \\
\quad \text{values (ID\_TEMPORAL\_TABLE\_sequence.nextval, NULL,} \\
\quad 1, ID\_table\_sequence.currval, NULL, NULL, sysdate); \\
\]

**7.3 Delete**

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; \( ID_{\text{tab}} \) now has the negative value.

<table>
<thead>
<tr>
<th>( ID_{\text{change}} )</th>
<th>( ID_{\text{previous_change}} )</th>
<th>( ID_{\text{tab}} )</th>
<th>( ID_{\text{orig}} )</th>
<th>( ID_{\text{column}} )</th>
<th>( BD )</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td>-1</td>
<td>A</td>
<td>NULL</td>
<td>11-3-2012</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>-2</td>
<td>E</td>
<td>NULL</td>
<td>11-9-2012</td>
</tr>
</tbody>
</table>

**Figure 13.** Example of the insert into temporal table after the operation delete

\[
\text{INSERT INTO temporal\_table} \\
\quad (ID_{\text{change}}, ID_{\text{previous\_change}}, \ID_{\text{tab}}, ID_{\text{orig}}, ID_{\text{column}}, BD) \\
\quad \text{values (ID\_TEMPORAL\_TABLE\_sequence.nextval, max\_change,} \\
\quad -1, : \text{old.ID}, NULL, NULL, sysdate); \\
\]

As we can see, the representation of the time is based on uni-temporal model, where attribute BD also determines the last time of the validity of the previous state. However, this representation cannot store unknown states of the objects. Thus, if the state is undefined the corresponding data about the object must be relocated into tab consisting of deleted objects. The principle of this method and also other implemented methods is described in [8] [9].

Proposed solution is fully temporal and although it has not been strictly written, this structure allows you to manage future valid data, too. One possible way to provide it is just functionality Job, by which is possible to plan changes of individual attributes. Functionality Job ensures automatic change of values and it is designed to handle methods, respectively provides executing the script in defined time. Triggers are connected to the Insert, Update and Delete operations, so the information about the change is
automatically stored in the temporal table. The planning of this event requires several parameters, the main ones include job_type - in our case it is a PL / SQL block, job_action characterizing what is going to be done. The other important parameter is start_time defining the time at which the job is planned to execute the PL / SQL block. After executing the body, the job should be cancelled respectively deactivated. Although the database would not be modified, because the process of change always check the columns whose attributes are changed, this could significantly affect the performance of the system.

Disadvantage of this system is the fact that it is not possible to get data valid in the future before the executing the Job, because the new values are not directly available. System ensures executing the script at specified the time, but information about future changes (although they are available in the system), cannot be obtained directly, although the new values can be found in the system tables. Problem is the structure of these records.

However, there is much more principal complication, the principle of which is shown in the figure 14. Suppose that at time $T_1$ an attribute $A$ of the object $O$ is planned to be changed to value $V_1$. Later, however, we find that the correct value of the attribute $A$ of object $O$ should be $V_2$. If the new time ($T_2$) change is equal or greater than $T_1$, the problem does not occur. However, if it is a change whose validity begins before time $T_1$, the problem does not occur. Otherwise we would get incorrect data from the time $T_1$, respectively it would be necessary to ensure that at the time $T_1$, incorrect state of the object would be modified. Therefore, it is necessary to store the information about the planned and scheduled Jobs, so they can be deactivated.

8 COMPLEX FUTURE VALID DATA PROCESSING

It has already been partially mentioned in the previous section, future valid data management brings a lot of facts and complications that must be considered during the project development. Whereas it is necessary to have data about changes in the future, table containing these future valid data was created. Specifically, it contains the value of the primary key - the object identifier (id_orig), type of operation (insert, delete, update) and the new attribute values (insert and update operations). However, this would not solve the problem mentioned before - updating values over time (fig.14). It is therefore necessary to save the identifier of the process, which provides the executing of the DML operation. - in our case it can be used the job_name. In essence, we have two possible choices, where to store the identifier, features and benefits will be described in the next section of this chapter.

9 IDENTIFIER OF THE JOB IN THE FUTURE TAB
The one possibility is to store the information about the job in the temporal table (fig. 16). The advantage is the access to all jobs, which were processed or will be processed in the future. However, not all transactions are executed using the \textit{Job} and thus, the column would contain a large number of undefined (\textit{NULL}) values. The question is also whether it is necessary to record this information after job processing and executing the body.

![Figure 16. Example of temporal table structure](image)

The second solution is to store the identifier of the \textit{Job} in the future table - table that contains information about the data valid in the future (fig. 17). After executing operation defined in the \textit{Job}, this information is deleted from the future table and therefore table only contains information about the actual planned activities. If there is a request to change the object state (insert, delete, update), system checks, whether there is any scheduled change of this object. If so, the user is notified and must choose – retention or cancellation of other \textit{Jobs} (or the parts of them) based on this object. The proposed solution is appropriate and meets the requirements comprehensively and simultaneously solves the mentioned problem.

![Figure 17. Structure of future table containing job_name](image)

### 10 SUMMARY

Complete structure of the fully temporal model can look like the figure 18. The model consists of the classical non-temporal tables. Future data are stored in future table till the execution or cancelling the process. Temporal table consists the reference to historical values, so any state during the life-cycle can be reconstructed.

![Figure 18. Structure of fully temporal model](image)

### 11 EXPERIMENTS

The proposed models described before were compared each other to declare the quality of the models based on the required processing time to get the state of the object or the snapshot of the whole database table. The table 4 shows the total number of the required tables and the processing time. The number of the records is 10 000 and the experiments were provided using the Oracle 11g database.

Three temporal models are compared - the first is the temporal model used for storing mostly
historical and current valid data. The other ones are fully temporal; the first one stores the information about the job in the temporal table and the last one stores these data in the future table. As we can see, the best choice for the fully temporal model is the last solution – fig. 20.

Table 4. Experiments results

<table>
<thead>
<tr>
<th></th>
<th>Temporal table (historical data processing)</th>
<th>Temporal table (full – job info in temporal table)</th>
<th>Temporal table (full – job info in future tables)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of temporal</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>columns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of conventional tables (containing temporal columns)</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>Number of future tables</td>
<td>0</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>Number of historical tables</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Number of temporal tables</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Size of the DB (kB)</td>
<td>18 093</td>
<td>19 874</td>
<td>18650</td>
</tr>
<tr>
<td>Time to get all actual data (current snapshot) (ms)</td>
<td>931</td>
<td>935</td>
<td>931</td>
</tr>
</tbody>
</table>

In comparison with partially temporal model - first one - the slowdown of the processing time is minor.

The overall slowdown of the third model in comparison with the first model is:

- **Size**: 3%.
- **Time to get current snapshot**: 0%.

As it was already mentioned, the results of the experiments can be considered satisfactory, because of the storing the entire history and also future valid data of temporal columns in the third model (fig. 20).

12 TABLE LEVEL VS. COLUMN LEVEL

In the systems operating with various temporal data related objects often appears the situation that the state is changed updating only one attribute. When using temporal table at the table level, the whole state must be updated regardless the number of the unchanged attribute values. However, our developed system based on temporal column (not temporal state) processing does not have that disadvantage. In addition, one more important problem can occur. Imagine the situation, that one attribute changes its value, but the new one is undefined. What is the problem? The problem is, that in standard model, the whole state is undefined and must be somehow processed, whereas, our system invalidates only one particular attribute, but the state of the object at the global level remains valid. Thus, our developed system can be called “Extended temporal system based on column temporal data processing”.

13 CONCLUSION

Developers nowadays require not only access to current valid data, but also require having overview of properties, structures and values at any time point or time interval. However, conventional database does not provide sufficient methods to ensure this requirement, therefore, the temporal system at level tables has been created, because of the management of backups and log files was too complicated and required large amount of time to obtain the necessary data. The aim of our development is to create a solution for managing objects throughout their life cycle. Our suggested solution also allows you to store the information about the objects after their logical deletion, if necessary.

This paper deals with the concept of temporal database - column level management, describes the properties and characteristics of the implementation and also shows their disadvantages. Consequently, the systems are compared on the basis of performance - the time required to obtain the required data as well as the size of the database, although the size is not the

Figure 20. Complete fully temporal model
primary criterion because of the price of hardware, today.

The proposed solution offers wide range of opportunities and is fully temporal and therefore usable in any area of life like simulation, optimization of processes, medicine, industry and so on. Simply in any area, where it is necessary to process and store the information over the time.

In the future, we are going to focus on the indexing structures for more effective work with this concept. Extending this model with the transaction time would be useful, too.

ACKNOWLEDGEMENT

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.

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