Searching and indexing in temporal databases

Michal Kvet*
Department of Informatics
Faculty of Management Science and Informatics
University of Žilina
Univerzitná 8215/1, 010 26 Žilina, Slovakia
michal.kvet@fri.uniza.sk

Abstract

This paper deals with the design and administration of complex temporal system based on attribute granularity. In the past, historical data were provided using backups and logs. Later, object level temporal system has been developed, which can produce many duplicate values if the change frequency and granularity is not the same for all object attributes. Proposed column level temporal architecture does not decrease the performance, if the attribute granularity differs. An important aspect of the temporal system is based on states selecting and changes of the states over the time monitoring. Several index structures have been defined, the performance of them based on access methods have been experimentally verified. Moreover, temporal concept of Select statement with new subelements has been defined. It was also necessary to define transaction management for conflict states resolution - each object can be defined by no more than one state at any timepoint. We have implemented access rules as well as hierarchy of priorities designed to address time collisions. In conclusion, we describe temporal management response to the change of data types, characteristics or data model extension. Classification rules for temporal models and access methods have been proposed.

Categories and Subject Descriptors

H.2 [**Database Manager**]

Keywords

Temporal system, column level approach,
index structures, access methods,
Select statement, transactions, database integrity

1. Introduction

Massive development of data processing has brought the need for large data access, which must ensure that all information are provided at the right time. Database systems are the core of almost every information systems and are one of the most important parts of the information technology. They are used not only in standard applications, but can be found also in critical applications like transport systems, industrial control systems, health care systems or technological process monitoring systems [1].
Most of the processed data represent current states of the objects. However, these states evolve and change their status over the time. Once the properties are changed, corresponding attributes are updated and the database still contains only actual valid data. In the past, historical data were provided using log files and backups. These structures stored also non-temporal data in the unsuitable form, which often resulted in operation loss. Later, the *Flashback* functionality has been proposed, which allows obtaining historical data table image. Monitoring and managing process of the changes and evolution over the time was very complicated and time consuming using that method. Moreover, paradigm of temporal management is based on the whole time spectrum, thus it should be possible to manage also future valid data. Using backups and log files, it was absolutely impossible to meet this requirement. Temporal system must also be able to cope with the requirement to change the data model, data types of the attributes or conversion from conventional attribute to temporal or vice versa. Temporal column is defined by the need to track individual changes (updates) over the time, whereas conventional attributes store only actual valid data.
Object level temporal approaches extend the paradigm of conventional databases by adding validity spectrum to the definition. Uni-temporal system uses validity, bi-temporal concepts are based also on transaction validity. In general, we can deal with the multi-temporal systems processing multiple time spectra and zones. Overall costs, time processing requirements, system response are the critical factors. As shown in the chapter 4, this solution does not cover all the problems of the processing and data management, therefore new model has been proposed allowing complex data attributes management. This paper can be divided into three parts. The first deals with the existing solutions, the second one describes developed architecture, which is performance evaluated in the third part.

2. Temporal system requirements

Temporal system requirements defined in [2] are designed for easy data object manipulation by the users. In our

*Recommended by thesis supervisor: Prof. Ing. Karol Matiaško, PhD.
Defended at Faculty of Management Science and Informatics, University of Žilina, August 2015.

opinion, these criteria do not cover the temporal complexity. Based on the results of the analysis, temporal aspects has been extended by us (aspect 3 and 4). Generally, requirements can be divided into these four groups [3, 4, 5]:

1. Aspect of usability is based on the easy manageable methods. The aim is to provide easy access to the latest results, as well as the results at any time in the past. Moreover, it should be possible to get changes based on the historical data and make projections for the future.
2. Aspect of performance is based on the correctness of the results. Time management and results should be the same in performance rate and form when accessing actual or historical data.
3. Aspect of data structure is a special requirement for storage optimization based on the relevance definition. Only values in the monitored area are necessary to be stored. If the attribute does not change its value or the change is not significant, there is no reason for storing and processing new value. System focuses therefore only on the objects of interest, positions, where the changed segment can be located. Therefore, the Epsilon (ϵ) value can be defined, which expresses the minimal change of values in the defined area, which should be stored. If the value $\epsilon = 0$, all the measured data are stored in the database. The principle is based on threshold.
4. Transaction management is based on transaction processing and definition. In the case of sensor data processing, this aspect can be replaced with the error detection and measurement corrections, if necessary.

Based on the requirements, several aims have been defined - design, implementation and semantics definition of the attribute temporal approach with the emphasis on undefined states and future events handling, design and implementation of the temporal integrity constraints and index definition and tablespace location management.

3. Object level uni-temporal system

The easiest and also often used method to manage temporal data is uni-temporal system. It is based on the extension of the conventional (non-timed) model. The primary key now contains not only the object identifier, but also one or two attributes determining the validity of the row. Consequently, one object can be defined by the various numbers of the rows, but not more than one defines the object at any time point. Thus, the data modelling operations must define not only the object itself, but also the timepoint expressing the begin timestamp (or other timed attribute based on the granularity like day, month and so on) or two attributes expressing the time interval. In our case, the row is defined by the validity. The figure 1 shows the structure of the uni-temporal system based on the validity interval.

Time interval can be represented by four ways - open-open, open-closed representation - which are used very rarely because of the problem of specifying validity beginning directly. Another two types are closed-closed and closed-open representation. The advantage of closed-open

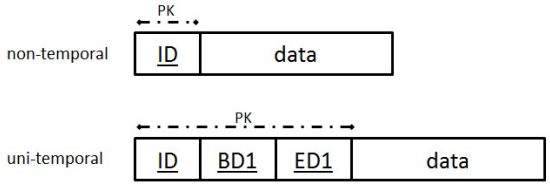

Figure 1: Conventional and uni-temporal structure [2]

representation is based on granularity change, which does not generate undefined states (gap). If the validity is not right side bordered, $Null$ value is not used because of the primary key definition. Moreover, using this clause, it is not possible to determine whether an event has occurred or not. Istead of $Null$, $MaxValueTime$ notation is used [4].
However, the validity can be modelled by a single attribute that expresses begin date. The next state delimits the validity of the previous state of the particular object. Our developed approach uses this principle. Moreover, it is necessary to create definition for invalid states [5].

4. Column level temporal model

The solution described in the previous section manages the attributes of the objects, whereas the standard uni-temporal model works with the whole objects. Column level temporal system is based on object attribute as the main part of the granularity. Thus, it is not problem, if the attribute granularity of the changes is not the same. Moreover, this system can manage objects with conventional attributes. The principles and described structure can be found in [3, 5].

5. Extended column level

Extended column level temporal system can be considered as the improvement of the column level temporal system in the terms of the performance and the simplicity of the model management for the users. It is extended by the definition of the type of the operation. If the operation is update, there is also the reference to the data type tables with historical values.
Existing applications are connected to the conventional layer with actual values, thus 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 developed *temporal_table* and historical value is inserted into to the table containing historical values. The figure 2 shows the complete structural model. Application is directly connected to the main tables with current valid values. It means that currently used applications can be used without any change. Historical values are stored in the special section, each temporal data type has one table defined by the identifier (primary key) got using the sequence and trigger and the values themselves. Thus, the principle and system is similar to the column level temporal system, but historical values management and temporal table is different.

Management of the temporal table is completely different. It consists of these attributes (fig. 3):

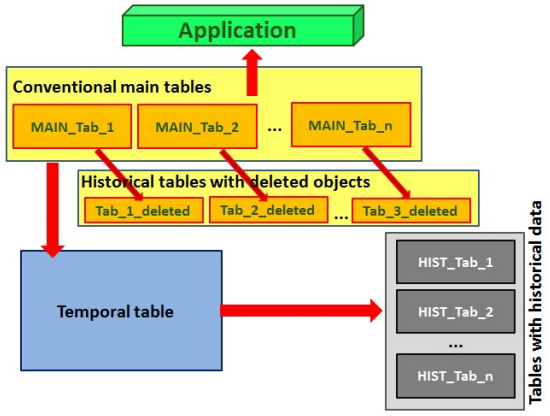

Figure 2: Extended column level temporal system structure

- *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:
  - $C = char/varchar$
  - $N = numericvalues(real, integer, \dots)$
  - $D = date$
  - $T = timestamp$

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

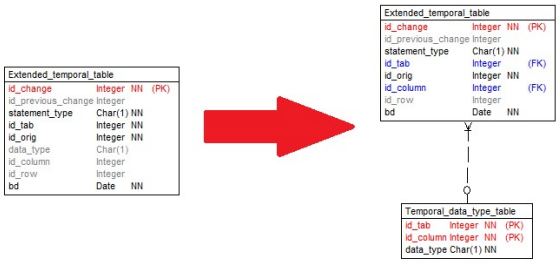

Figure 3: Extended_temporal_table

The principles of data operation modelling and management are described in [6].
One of the temporal property is the definition of the volatility. Often, it is not necessary (and even useful) to store states during indefinite time period. Therefore, proposed solution contains algorithms and methodology for database reconstruction after too historical data removal to aim the requirement of consistency. The method parameter can be defined by the timepoint or by number of following changes [7, 8].
The object state in real time can be defined by no more than one valid state. However, it may cause collisions during the new conditions definition, therefore, we expanded transaction manager by the access rules (*restricted*, *partial*, *complete*, and *warning*) and the definition of transaction priority. If the system of the priority is used, transaction with higher value can directly influence the transaction with lower value of the priority. Access rules delimits the planned events.
The basic unit of a database system is a transaction which is in conventional approach characterized by four basic properties - atomicity, persistence, consistency and isolation. The existing definition of transactional consistency has been extended by the temporal entity integrity and referential integrity strict rule. This rule requires that the referenced object must completely cover the validity of the particular table row.
Object properties may be additionally defined by the time records in several tables. Therefore, we have created a methodology managing hierarchically modelled objects in the system (definition of temporal ISA hierarchy). We have also defined approaches and techniques allowing and managing changes of a data model in time.

6. Temporal Select definition

The *Select* statement in relational database approach is considered as the most important and most frequently used SQL statement based on performance. With this statement, we get desired data from the database using relational tables. The basic syntax of the *Select* statement in conventional database consists of these six parts - *Select*, *From*, *Where*, *Group by*, *Having* and *Order by*.
Although conditions can be defined in the *Where* clause, this segment does not cover the complexity and structure of the temporal system. Therefore, the following section describes ways to enhance the whole concept of management of temporal data.
Designed and implemented syntax shows the temporal extension of the *Select* statement using these parts:

- EVENT_DEFINITION,
- EPSILON_DEFINITION,

- MONITORED_COLUMN_LIST,
- TYPE_OF_GRANULARITY.

6.1 *Event_definition*

Clause *Event_definition* extends the *Where* clause of the *Select* statement and specifies processed time range by a point in time (*defined_timepoint*) or by time interval modelled by the closed-closed or closed-open representation (all defined types can be transformed to another types [2]) - fig. 4.

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

Figure 4: Event_definition

6.2 *Epsilon_definition*

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

6.3 *Monitored_column_list*

The clause *Monitored_column_list* as the extension of the *Select* for temporal approach allows list of columns definition, which are relevant for processing and should be monitored. This list does not need to be identical to the first part of the *Select* statement, however, it can consists only of the temporal attributes (not conventional or functions).

6.4 *Type_of_granularity*

The clause *Type_of_granularity* defines the format of the output set - all attributes, new attribute values or actual and previous state definition. As it has been already mentioned, the core of the system consists of the temporal analysis of changing characteristics.

7. Index structures

One of the main features of optimization is based on index structures. Temporal databases are oriented for state management and monitoring over the time. Getting states and individual changes in the *Select* statement form the core of the 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. If a heap-organized table has no indexes, then the database must perform a full table scan to find a value.

7.1 *Index sequence file*

Index sequence data alignment is based on two data sets - the sequential file and the index file, consisting of a key and a pointer to the data row in a sequential file. Finding the record is based on the index scan and access to primary (sequential) file. If the file was too large, it is possible to add another index layer - hierarchical index. The main disadvantage of index-sequential arrangement is the significant performance decrease in performance (processing time requirements and the response to it) based on amount of data increase. This problem can be partially solved by the data index reorganization.

7.2 *B-tree, B+tree*

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.
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).
B+tree extends the concept of B-tree 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 (fig. 5).

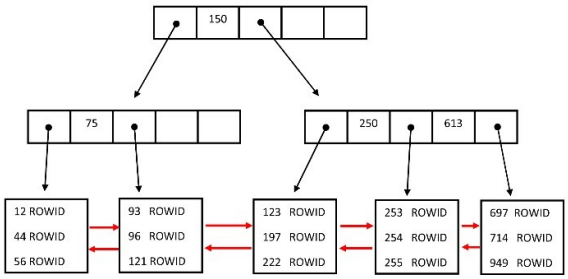

Figure 5: B+tree

Limitation of this approach is a small number of records (low cardinality). In that case, using index does not produce the desired effect in terms of performance (acceleration). Another disadvantage is the lack of support SQL queries with functions implicitly.

7.3 *Inverted key B-tree*

Index B-tree structure with inverted key is used in case of often requirement for tree balancing (column value is obtained using the sequence and the trigger - autoincrement) caused by frequent execution of the *Insert* statement. Indexing will not use original key value, but the inverted variant. For example, for the key 123 is inverted key value 321.

7.4 *Bitmap index*

Bitmap indexes are represented by two-dimensional array, the number of rows is identical to the cardinality of the table. The first column contains a reference to a record in the data file, the other columns are called bitmap and represent different values of the indexed column.

The following figure illustrates the general assessment of conditions of the order using bitmap indexes. Let have a table - cars with primary key represented by the registration number (*license_plate*). Non-key attributes include color (*color*), producer (*producer*), construction year (*year*) and price (*price*). The aim is to find red cars produced by the "Škoda" brand in 2012. The solution is shown in fig. 6.

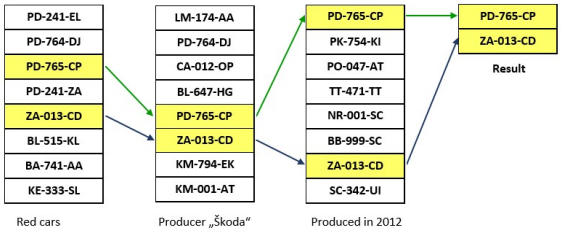

Figure 6: Bitmap

7.5 *Table index*

If the table is too small, it is better to keep complete records directly instead of references (pointers) to the leaf level data. This approach reduces the number of Read operations and thus accelerates the evaluation of the request. Oracle (version 8 and later) allows you to create a special type of table ($index-only table$), where the entire table is stored in the index structure (B+tree).

7.6 *Cluster index*

Cluster index provides the physical layout and location of the data set by attributes constituting the index. Accordingly, each table can have no more than one index of such type.

7.7 *Table cluster*

Cluster tables is linked to several tables, records with the same key value (*clusterkey*) are stored together - these data are usually required together. It means, that the clustered data can be loaded using one *Read* operation. In temporal system, historical and future valid data are clustered together with actual data to provide complete monitoring. Cluster key is the identifier of the data extended by the definition of the table, if the primary key identifier does not provide it automatically.

8. Tablespace localization

Performance characteristics are influenced by the index location and by the type of the index structure. In the previous section, several index types has been described. However, second criterion is based on data division into separate tablespaces, which are located directly with the data themselves or in the separate discs, which allows more flexible response to the extension of the number of data blocks and index construction changes. The last type of the processed type is based on index localization in the remote storage (server) and access using the network. The aim is to monitor the network traffic and response of the system in case of network failures and routing. In this case, if using 1Gb network speed, the slowdown of the system varies usually from 5% to 10%.

9. Summary

Temporal data processing and complex management is one of the most important factor of the current database systems. Conventional database approach is not suitable for progress monitoring over the time. Data processing requires access to the whole information about the evolution of the states during the life-cycle. Effective managing temporal data is the core of the development and can be used for decision making, analyses, process optimization, which is very significant factor in industrial environment. During the analysis, we found out, that there is currently no complex temporal system classification, therefore we proposed, presented and discussed classification criteria consisting four-layer architecture - the type of database system, the type of temporal structure, type of transaction processing and the type of index.
Temporal data searching and monitoring is the fundamental part of the processing using *Select* statement. Current conventional implementation does not cover temporal architecture, therefore we expanded the definition with the several clauses - processed period and granularity, monitored temporal attributes and epsilon definition.
Temporal characteristics are based on object level, which can in many cases generate duplicates, therefore column level model has been defined, implemented and tested with emphasis on transaction management.
Temporal data are usually extensive, thus it is necessary to manage them effectively in terms of memory requirements and overall management and response of the system. Over the following proposed attribute oriented model, we created different variants of index structures, which make it possible to get desired results much faster. The final part of the experimental work is the comparison of the proposed indexes with regards on their location, access methods, and defined rules.
In the future, we would like to focus on temporal table clustering and index division to the network nodes based on multiple characteristics.

Acknowledgements. 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,
Centre of excellence for systems and services of intelligent transport II, ITMS 26220120050 supported by the Research & Development Operational Programme funded by the ERDF,
Center of translational medicine, ITMS 26220220021 supported by the Research & Development Operational Programme funded by the ERDF.
Podporujeme výskumné aktivity na Slovensku - projekt je spolufinancovaný zo zdrojov EÚ."

References

[1] Date, C.: *Date on Database*, Apress, 2006.
[2] Johnston, T. et al.: *Managing Time in Relational Databases*, Morgan Kaufmann, 2010.
[3] Kvet, M., Matiaško, K.: *Epsilon temporal data in MRI results processing*, In Proceedings of the 10th international conference - Digital Technologies, 9.6. - 11.6.2014, Žilina, 2014.
[4] Kvet, M., Matiaško, K.: *Transaction management in temporal system*, In Sistemas y Technologías de Información: actas de la 9a conferencia Ibérica de

Sistemas y Technologías de Información, 18.6. - 21.6.2014, Barcelona, 2014.

[5] Kvet, M., Matiaško, K.: *Column level uni-temporal data*, In Communications: scientific letters of the University of Žilina, Volume 16, Issue 1, 2014.

[6] Kvet, M., Vajsová, M.: *Extended column level temporal system indexing*, In UKSim-AMSS 8th European modelling symposium on computer modelling and simulation, 21.10. - 23.10.2014, Pisa, 2014.

[7] Maté, J.: *Transformation of Relational Databases to Transaction-Time Temporal Databases*, In Engineering of Computer Based Systems (ECBS-EERC), 2011 2nd Eastern European Regional Conference, 2011.

[8] Snodgrass, R. et al.: *Adding Transaction Time to SQL/Temporal*, 1996.

Selected Papers by the Author

M. Kvet. *Temporal data approach performance* In New developments in Circuits, systems, signal processing, communications and computers: proceedings of the international conference Circuits, systems, signal processing, communications and computers (CSSCC 2015): Vienna, Austria, March 15-17, 2015. pp. 75-83

M. Kvet, M. Vajsová. *Extended column level temporal system indexing* In EMS 2014, 21-23 October 2014, Pisa, Italy, 2014. pp. 5-10

M. Kvet, K. Matiaško. *Transaction management in temporal system* In Sistemas y Technologías de Información: actas de la 9a conferencia Ibérica de Sistemas y Technologías de Información, 18.6. - 21.6.2014, Barcelona, 2014

M. Kvet, K. Matiaško, M. Kvet. *Transaction management in fully temporal system* In UKSim-AMSS 16th international conference on computer modelling and simulation, 26-28 March 2014 Cambridge, United Kingdom, 2014. pp. 147-152

M. Kvet, K. Matiaško. *Uni-temporal modelling extension at the object vs. attribute level* In EMS 2013, Manchester, United Kingdom - 20-22 November 2013. pp. 6-11

M. Kvet, A. Lieskovský, K. Matiaško. *Temporal data modelling: conventional and temporal table* In ICCSE 2013, April 26-28 2013, Colombo, Sri Lanka, 2013. pp. 452-459