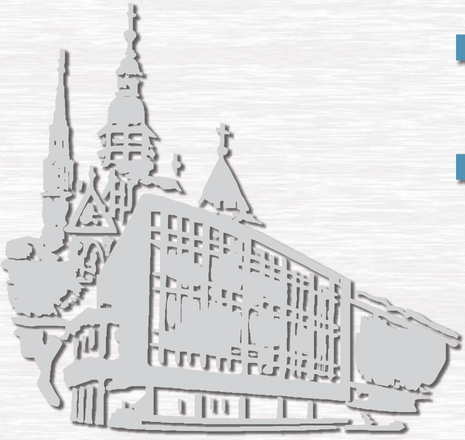

ICETA 2022

International Conference

20th Anniversary of IEEE International Conference on Emerging eLearning Technologies and Applications

IEEE
Czechoslovakia Section

PROCEEDINGS

Information and Communication Technologies in Learning

October 20-21, 2022
Grand Hotel Starý Smokovec, High Tatras
Slovakia

ICETA 2022

20th Anniversary of IEEE International Conference on Emerging eLearning Technologies and Applications

PROCEEDINGS

October 20 – 21, 2022

Grand Hotel Starý Smokovec, High Tatras, Slovakia

Website: http://www.iceta.sk

ISBN: 979-8-3503-2032-9
IEEE Catalog number: CFP2238M-USB
Proceedings editor: František Jakab

COMMITTEES

Honorary Committee Chair

Kmeť Stanislav, Technical University of Košice

Honorary Committee

Molnár Ľudovít, Slovak University of Technology, Bratislava
Newman B. Harvey, California Institute of Technology, Pasadena
Rudas Imre, Óbuda University, Budapest

Program Committee Chair

Jakab František, Technical University of Košice

Program Committee

Babič František, Technical University of Košice
Bauer Pavol, Delft University of Technology
Bauerová Danuše, Technical University, Ostrava
Behun Marcel, Technical University of Kosice
Berke József, John von Neumann Computer Society, Budapest
Bieliková Mária, Slovak University of Technology, Bratislava
Brestenská Beáta, Comenius University Bratislava
Cehlar Michal, Technical University of Košice
Chovanec Martin, Technical University of Kosice
Čičák Pavol, Slovak University of Technology, Bratislava
Čižmár Anton, Technical University of Kosice
Dado Milan, University of Žilina
Doboš Ľubomír, Technical University of Košice
Dolnák Ivan, University of Žilina
Doroshenko Anatoliy, Institute of Software Systems of NASU, Kiev
Drozdová Matilda, University of Žilina
Dzupka Peter, Technical University of Košice
Feciľak Peter, Technical University of Kosice
Galgoci Gabriel, AT&T Bratislava
Gavurova Beata, Technical University of Košice
Genči Ján, Technical University of Košice
Haľama Marek, Technical University of Košice
Hämäläinen Timo, University of Jyväskylä
Hautamaki Jari, University of Appl. Science
Hluchý Ladislav, Slovak Academy of Sciences, Institute of Informatics
Horváth Pavol, SANET - Slovak Academic Network, Bratislava
Huba Mikuláš, Slovak University of Technology, Bratislava
Hudak Radoslav, Technical University of Košice
Hvorecký Jozef, Vysoká škola technická a ekonomická in České Budejovice
Jelemenská Katarína, Slovak University of Technology, Bratislava

Juhár Jozef, Technical University of Košice
Juhas Gabriel, Slovak University of Technology in Bratislava
Kainz Ondrej, Technical University of Košice
Kess Pekka, University of Oulu
Kireš Marian, University of Pavol Jozef Šafárik
Klačková Ivana, University of Žilina
Klimo Martin, University of Žilina
Korshunov Alexander, Kalashnikov Izhevsk State Technical University of the name M.T. Kalashnikov
Kotuliak Ivan, Slovak University of Technology
Kováčiková Tatiana, COST Office, Brusel
Kovács Levente, Óbuda University, Budapest
Lavrin Anton, Technical University of Kosice
Lelovsky Mario, Slovak IT Association Bratislava
Levický Dušan, Technical University of Košice
Lumnitzer Ervin, Technical University of Kosice
Mesaroš Peter, Technical University of Košice
Meško Dušan, Jessenius Faculty of Medicine, Martin
Michalko Miroslav, Technical University of Kosice
Modrak Vladimir, Technical University of Kosice
Mytnyk Mykola, Ternopil National Ivan Pul`uj Technical University
Nakano Hiroshi, Kumamoto University
Pavlik Tomaš, Technical University of Košice
Pietrikova Alena, Technical University of Kosice
Pisutova Katarina, Comenius University
Piteľ Jan, Technical University of Košice
Poruban Jaroslav, Technical University of Košice
Radács László, University of Miskolc
Restivo Maria Teresa, University of Porto
Ristvej Jozef, University of Zilina
Salem Abdel-Badeeh M., Ain Shams University of Cairo
Semanišin Gabriel, University of Pavol Jozef Šafárik
Simonics István, Obuda University, Budapest
Šimšík Dušan, Technical University of Košice
Sinčák Peter, Technical University of Košice
Sobota Branislav, Technical University of Košice
Sovak Pavol, University of Pavol Jozef Šafárik
Stopjakova Viera, Slovak University of Technology in Bratislava
Strémy Maximilián, Slovak University of Technology in Bratislava - Advanced Technologies Research Institute – University Science Park CAMBO, Trnava
Stuchlikova Lubica, Slovak University of Technology
Šveda Dušan, UPJŠ Košice
Szabo Stanislav, Technical University of Košice
Szentirmai László, University of Miskolc
Turna Jan, Comenius University, Bratislava
Usawaga Tsuyoshi, Kumamoto University
Vagan Terziyan, University of Jyvaskyla
Vasková Iveta, Technical University of Košice

Vokorokos Liberios, Technical University of Košice
White Bebo, SLAC National Accelerator Laboratory, Stanford University
Zivcak Jozef, Technical University of Košice
Zolotová Iveta, Technical University of Košice

Organizing Committee Chair

Jakab František, Technical University of Košice

Organizing Committee

Fejedelem Štefan, elfa, s.r.o., Košice, Conference Manager
Rakoci František, elfa, s.r.o., Košice, Webmaster
Zámečníková Iveta, elfa, s.r.o., Košice, Finance Chair

TABLE OF CONTENTS

Concept of Hybrid Temporal Architecture

Michal Kvet [*]
[*] Žilinská univerzita v Žiline, FRI, Žilina, Slovakia
Michal.Kvet@uniza.sk

***Abstract*—Temporal databases were introduced almost immediately after the first database systems were released. They encapsulate the state by the temporal frames, mostly expressing validity. Three temporal architectures reflecting the granularity perspective are currently used – object, attribute, and synchronization group. These approaches provide sophisticated solutions for any temporal system. However, they are not robust, if mixed temporal insights need to be modeled. The proposed hybrid temporal architecture focuses not only on modeling temporality but also conventional and static data approaches can be incorporated by providing various temporal commands regarding processing and registration.**

I. INTRODUCTION

Database technology forms an inseparable part of the current information systems by separating the application and data layer. Data themselves can be modeled using various architectures, based on their structures, origins, reliability, and precision, supervised by the data distribution techniques to ensure availability, if the instance, hardware, or connectivity fails. The most often used principles are associated with the relational paradigm, object-oriented or object-relational form, or NoSQL pointing to Big data perspectives.

Transaction support is a core element of the database technology ensuring any data tuple passes the defined rules forming the data integrity and consistency. Database integrity definition is encapsulated by these five categories. Entity integrity is a critical part of data modeling by enabling data table relationships, reflecting the primary key, which should be unique and minimal. Thus, any subset of the primary key attribute list does not cover the uniqueness definition. Referential integrity relates to the foreign key, which can hold the referenced primary key (or unique constraint) value or NULL if a particular constraint allows it. Domain integrity is made of the data type for each attribute forming the list of values, which can be present, getting the orderability and comparison option. A domain can be enhanced by column integrity dealing with undefined values and unique constraints. Finally, user integrity encapsulates additional data requirements originating from the application domain. The transaction itself is a main processing unit of the database system and shifts the processing from one consistent state to another one. Thus, at the end of the transaction, all integrity rules must be passed [1].

The transaction is delimited by the properties of atomicity, consistency, isolation, and durability, meaning, that the whole transaction is either approved or refused completely. It is not acceptable to make only part of the transaction. Isolation and durability options relate partially to temporal management. It must be ensured, that only approved data changes are obtained during the querying and processing. Thus, before approving the transaction by spreading the changes, it is not acceptable to obtain such data out of that transaction. It requires storing original values, respectively the original values should be able to be obtained or reverted. Similarly, after committing the transaction, durability must be ensured, even after the collapse of the data server or infrastructure. Atomicity, consistency, and isolation perspectives can be directly covered by the database technology itself, the durability aspect requires multiple layer coordination, as well as ensuring the concepts on the hardware and storage layer [1].

Activities done inside the transaction are logged. Logs describe the activities inside the transaction and can be divided into UNDO and REDO types, consisting of the original state, new state forming the change vector, executed operation, transaction identification, timing, and System Change Number (SCN) values characterizing the state of the database. UNDO part of the transaction logging is stored in the UNDO tablespace physically in the database and dynamically rewritten if particular data are not necessary later. REDO is temporarily stored in the instance memory by copying its content dynamically to the database storage. A transaction cannot be approved if all the REDO data part of that transaction are not physically stored. Therefore, the transaction approval is just referenced by the REDO log processing [2].

Although it may not be obvious at first glance, transaction logs have significantly contributed to the research and development of temporality. Namely, by applying change vectors, it is possible to get the historical image of the data. Generally, if all transaction logs are present in the system, any historical composition can be obtained, by extracting the change vectors relevant to the object. Archivelog mode is operated by enabling the Archiver background process, which copies the online logs before overwriting them. Online logs are formed by a circular list that maintains data about active transactions.

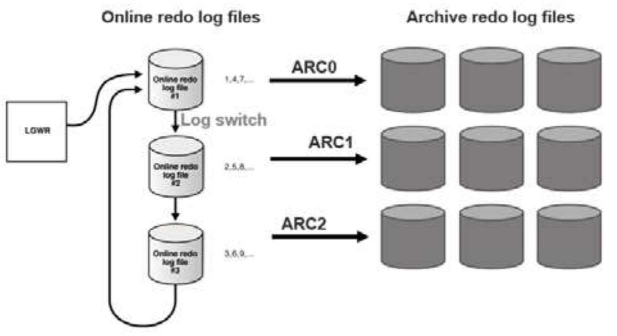

Figure 1. Archivelog mode architecture [2]

979-8-3503-2033-6/22/$31.00 ©2022 IEEE

Fig. 1 shows the Archivelog mode architecture by focusing on the Archive repository storing all data logs. Getting historical data can be automated using Flashback technology by enabling to revert of the whole database, table, or query output to the historical data image.

Flashback data management can be considered as the first temporal system by managing current valid data in a primary repository, however, historical data perspectives can be modeled and reconstructed using the logs. There are, however, several limitations related to temporality. Firstly, it is impossible to manage future valid data, just the history can be reflected, limiting the solution to the past. Secondly, if any transaction log were missing, reconstruction would be impossible, even if a particular data file did not process the required data in the query at all. The system cannot evaluate it, and, therefore, the entire operation would be rejected from the point of view of reliability. Last but not least, such a temporal solution would be too demanding in terms of time processing, technical possibilities, and operability. Flashback technology is primarily used to reconstruct the database after the failure, to get the historical data just once, it is not intended for large-scale dynamic processing.

As evident, temporal data management must be empowered by the physical models and architectures dealing with the states, collisions, validity, reliability, and transaction perspectives. Chapter 2 deals with the database architecture summary focusing on the data availability and reliability using database containerization. Chapter 3 deals with the temporal architectures from the granularity point of view. The proposed solution is aimed by Chapter 4 dealing with the hybrid temporal architecture.

Unlike existing systems, it does not focus only on the effective management of the temporal states of objects in time, it also reflects temporal registration, thanks to which the system can cover any temporal access, i.e. to store data whose changes do not need to be monitored, or data, which cannot be temporally processed for security, law restrictions and other reasons. In addition, such a solution can mark data sets for which it is not necessary to monitor changes and it is sufficient to store only the current state, or status valid in the future, respectively. And last but not least, it allows you to manage static attributes that do not change their value over time.

II. DATABASE ARCHITECTURE SUMMARY

Each database system is composed of two essential elements. Physical storage, file system, and various parameter files are delimited by the database. Multiple data files are commonly present in the database, formed by the set of blocks to be located. Block orientation ensures the finest granularity of the data to be transformed into the memory and evaluated. The size of the block is fixed, generally inherited from the database definition itself, however, each table can overwrite the value by using the tablespace as a separator between the instance and the database. A tablespace is a logical container covering the physical data files by getting the layer of abstraction.

Among the database as an essential component of the database system, there must be software for managing, manipulating, evaluating, and storing data, called the instance. An instance is a set of background processes manipulating the instance and data themselves. It is also formed by the memory structures for getting metadata, parsed and pre-processed execution plans, data in Buffer cache, as well as structures for the statistics, instance monitoring, etc. Fig. 2 shows the architecture of the database system reflecting the user interface. As evident, the user cannot directly operate the data, the whole activity is done by the background process supervision. Before connecting to the server, the client process is invoked, contacting the database interface – listener, which navigates the connection to the server process and maps the client to it. Consequently, the user can directly reflect the server process on the database system site. The common mapping between the client and server process is 1:1, forming the dedicated system. However, if the number of sessions is too high, it would be too demanding to create each server process separately, whereas it requires additional memory structure private for the session (PGA), holding the cursor state, local parameters, variables, etc. In that case, shared server architecture is used by using the dispatcher layer to navigate the user requirements to the fixed number of server processes [2] [3].

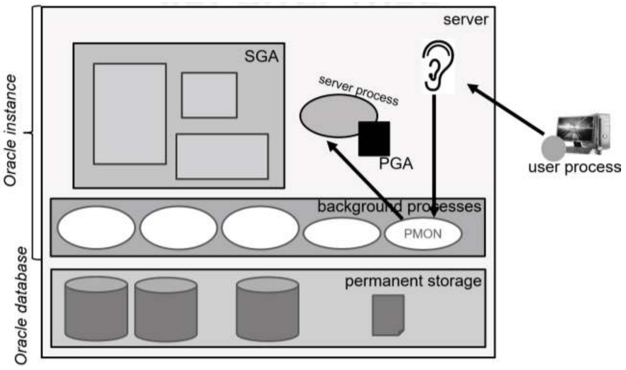

Figure 2. Server-client architecture

A. *Instance memory structures*

The required memory structures for the database instance are Buffer cache, Log buffer, and Shared pool. Optional structures include a Large pool, Java pool, and Streams pool.

The database Buffer cache is a working area for SQL statement execution, whereas the data are not updated directly in the database, instead, the changes and management are done in memory, supervised by the transaction logging. The database Buffer cache is delimited by the matrix of the blocks, which can be in a clean, dirty, or empty state.

The Log buffer is a short-term staging area for transaction management, holding change vectors before they are written to the Redo log files in the physical database storage. This structure, supervised by the Log Writer (LGWR) background process, guarantees no data loss from the transaction perspective. It can form the bottleneck of the whole system's performance if the size is not set properly.

A Shared pool is the most complex component of the Shared memory area. It consists of multiple subelements and substructures, like Library cache storing parsed form of the SQL queries of the recently executed code, Data dictionary cache manipulating the metadata, or PL/SQL area.

Large Pool is an optional area, mostly used for shared server architecture, storing backups, etc. Java pool is associated with the Java-stored procedure execution. Streams pool is a data transfer storage extension in case of using Data Stream architecture [2] [3] [4].

B. *Background processes*

Background processes are launched when the instance is started, or manual management can be used. There are dozens of processes forming the database system core, from the data management and performance perspectives, the following processes are inevitable.

System Monitor (SMON) ensures the housekeeping processes if the database is in an open state. The main aspect is related to the process of instance creation and interconnection between the database and instance.

Process Monitor (PMON) manages server processes by launching them, monitoring them, and terminating them. It also detects transaction failure after the session loss.

Database Writer (DBWn) is responsible for storing data by transferring them from the Buffer cache into the physical repository.

Log Writer (LGWn) operates the online transaction logs, managing UNDO and REDO structures. If the archiving is enabled, Archiver (ARCn) background process is enabled, copying the online redo log files after the log switching operation into the archive repository. Active REDO logs cannot be overwritten, whereas they hold transactions, which are currently present and have not been refused or approved, yet. The current log file is just that file that is being written to. Finally, the Inactive type holds data about the transactions, which are not required for the transaction recovery and can be overwritten. Naturally, if the Archivelog mode is used, it must be preceded by the Archiver background process operation.

The other processes are Recoverer, Dispatcher process, Deadlock detectors, Flashback data archiver process, and many more.

Fig. 2 shows the simplest database architecture by extracting the database and instance itself. Although transaction support always ensures data recoverability, if the instance or data layer fails, the whole database system becomes unavailable for serving requests. The shown database architecture is a single-tenant database using 1:1 mapping. Thus, each instance serves just one database.

To make the system error-prone and robust, it is inevitable to share the infrastructure across multiple nodes. Real Application Cluster (RAC) architecture manages multiple instances for one database allowing the spread and the workload balance. If any instance fails, the rest surviving ones can take the requests by the dynamic routing. Fig. 3 shows the architecture. It uses multiple instances, a user points to the RAC listener pointing the client to the local listeners, associated with each instance separately.

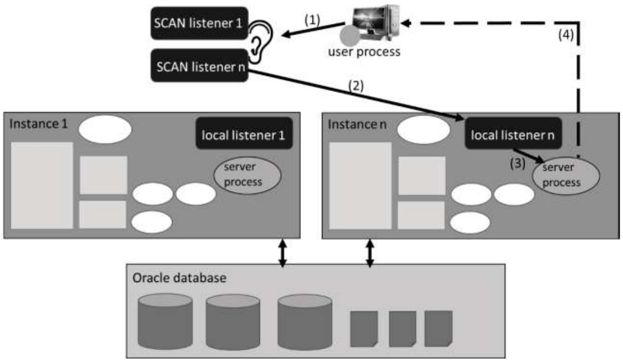

Figure 3. Single-tenant RAC architecture

The general solution is provided by the multi-tenancy. It is based on splitting the parameter files (control files, redo logs, metadata) managed by the root container and data files themselves operated by the pluggable databases. Thanks to those architectures, databases can be plugged and unplugged dynamically. Moreover, using the RAC option offers an ideal solution applicable to any system by providing reliability, durability, and almost permanent availability. Fig. 4 shows the multi-tenant container database. The instance layer is enhanced by the RAC.

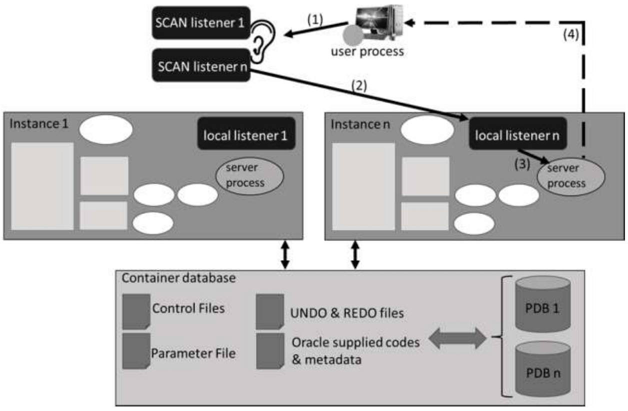

Figure 4. Multi-tenant RAC architecture

III. TEMPORAL ARCHITECTURES

A. *Temporal requirements*

To ensure the complexity, availability, reliability, and robustness of the information system, database architecture, access methods, as well as performance should be emphasized. In the previous chapter, advanced database architectures were discussed to highlight the availability, supervised by the transaction support ensuring the durability of the approved transactions, empowered by the transaction logs, and optionally by the archiving. This section summarizes existing temporal architectures reflecting the granularity. By using complex decision-making, it is inevitable to store not only current valid data, but the whole evolution should be reflected. The management points to the historical data, to reflect the changes and outputs of the previous decision. On the other hand, it is also necessary to emphasize future valid data, which get the future perspective and planned changes. And above all, current valid data access should be direct, straightforward, and as effective, as possible. Furthermore, current valid states should be extracted and provide the original outputs, as they existed without temporal support, whereas many existing applications use just conventional approaches and there is a huge requirement for accessing the data in a common way with no necessity to reconstruct and reflect existing systems.

B. *Object-level temporal system*

Object-level temporal architecture is based on the unique object identifier extension by the temporal frames, mostly expressing validity as a core element. However, generally, several temporal spheres can be used, as part of the primary key. Thus, the state itself is not just delimited by the object itself, time reference should be present, as well. Uni-temporal architecture is based on using one time sphere (validity), which allows you to sort the states in a timeline. The bi-temporal solution uses two time dimensions – validity and transaction time reference. Thanks to that, data corrections can be stored for each temporal state. Transaction reference can be modeled either by the sequence, but the time perspective is more reliable and powerful, whereas the data correction can be placed on the timeline.

Fig. 5 shows the architecture of the object-oriented approach.

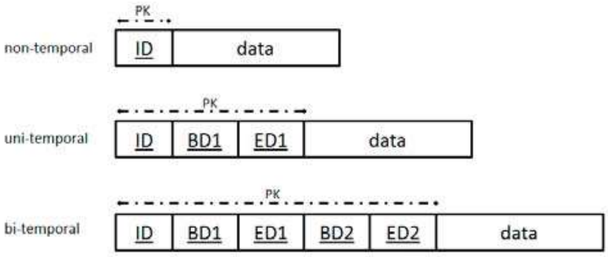

Figure 5. Object-oriented approach [3]

Temporal spheres can be modeled and represented using the time interval forming the begin (BD) and end (ED) point of the validity, characterized by various representations, mostly delimited by the closed-closed or closed-open approaches. Another approach is defined by storing only begin point of the validity. In that case, each consecutive state of the particular object limits the validity of the direct predecessor. Fig. 6 shows various time reflection modeling [5].

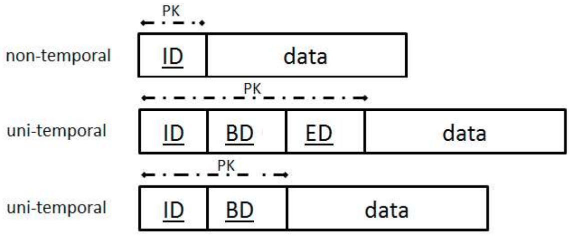

Figure 6. Time reflection modeling in an object-oriented approach [5]

C. *Attribute-oriented approach*

The attribute-oriented approach uses the finest granularity by emphasizing the attribute, meaning, that each attribute is separately enhanced by the validity time frame or other spheres. Thus, the state of the object is based on individual attribute reflection and composition. To ensure the performance and direct interconnection to the older applications, the current valid state layer is present, either physically or by using logical concepts via object views.

This architecture uses three layers and is shown in fig. 7. Current valid states are in the first layer, and historical and future valid perspectives are separated in the third layer. The most essential model is delimited by the temporal layer, which registers any change in a temporal table, which is formed by these attributes [5] [6] [7] [8]:

- *ID_change*
- *ID_previous_change* – references the last change of an object identified by *ID_orig*. This attribute can also have the *NULL* value which means, the data have not been updated yet, so the data were inserted for the first time in the 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 information about the identifier of the row that has been changed.
- *ID_column*, *ID_row* – hold the referential information to the old value of the attribute (if the DML statement was *UPDATE type*). Only update statement of temporal column sets *NOT NULL* value.
- *BD* – the begin date of the new state validity of an object.

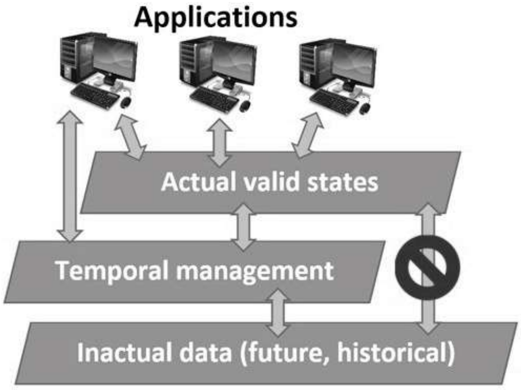

Figure 7. Attribute-oriented approach

Users can directly access only current valid states, all other activities are hidden and operated by the Temporal Manager background process ensuring consistency, robustness, reliability, and transaction reflection.

The extension of the attribute-oriented granularity has been introduced in [7]. Instead of using a separate historical data layer for each attribute, the non-current values are grouped based on the data types or data categories. Thanks to that, the number of data to be stored can be lowered by significantly lowering storage demands. The approaches, architecture, and layers remain the same compared to the attribute-oriented granularity.

D. *Synchronization group management*

The intersolution between the object and attribute granularity is provided by the synchronization groups. Each temporal group is temporally oriented and references the existing subgroup or individual attributes. Thanks to that, temporal layer demands are lowered, whereas the synchronization group is treated as one element, not individual attributes. The definition of the group is automated, reflecting the internal parameters to ensure performance. Its drop is also automated, optimizing the storage and references.

The solution is based on attribute granularity by extending the layers by group handlers. The architecture of the synchronization group temporal architecture is shown in fig. 8.

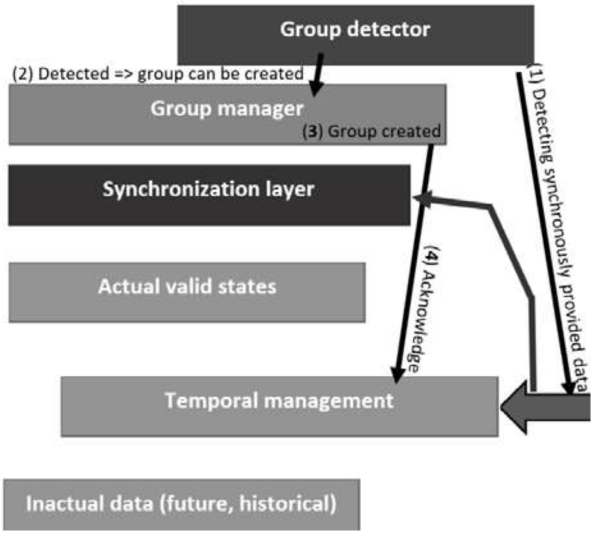

Figure 8. Synchronization group temporal model

The reference of the group from the temporal layer is done using the data_val pointer. The data model of the synchronization group management is in fig. 9.

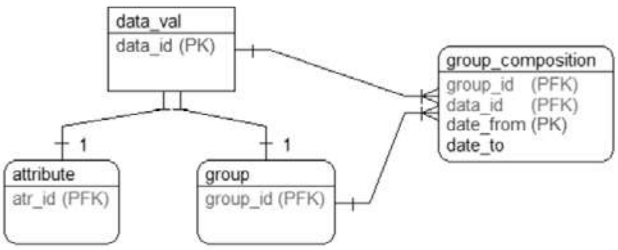

Figure 9. Data_val

IV. HYBRID TEMPORAL MODEL

The above architectures provide robust solutions for maintaining temporal data evolution. They are based on various granularity perspectives. The group-oriented solution is universal, whereas it can cover any precision – borders formed by the object and attribute perspectives can be also covered. The core element is however temporality itself. However, as evident, there can be various attributes enhanced by the additional restriction temporally. Namely, many attributes cannot process historical data due to security, law, or other circumstances. Similarly, historical values for some data sets do not be necessary to be stored, whereas they do not bring any additional information value. And finally, there can be multiple attributes, which do not evolve over time. So why reflect them by the temporal layer?

The hybrid temporal model proposed in this paper- extends the previous architectures by the temporal registration, providing the ability to process the data using any perspective. In general, there are seven available options – static, conventional, future, full_temporal, restricted, timing, count, and own_constraint.

The static option characterizes the data attribute sets, which do not evolve over time, typically delimited by the code lists or some init parameters. Thus, it is clear, that the data will not be changed, it is prohibited by the trigger or by using privileges, respectively.

Conventional attributes are not covered by temporality monitoring meaning, that the historical data are not stored and evaluated.

The Future option limits the historical management, which is not monitored, however, future valid data can be present and are autonomously transformed to the current states in terms of validity.

The Full_temporal option is a general solution managing data in a temporal manner, focusing on the history, as well as future valid data, optionally enhanced by the data corrections.

Historical data do not need to be stored unlimitedly permanently. Options Timing and Count reflect the usability of the historical values by forming the gate for the Delete operation. It is based on the number of consecutive newer states or the time perspectives can be defined, delimited by the Date or Timestamp precision. Thus, if the state is older than the defined limit, it is automatically deleted. For these options, two approaches can be present – strict and relaxed. If the Strict option is used, the delete or transfer operation must be done immediately after the conditions are met. Related type only marks such data and the Temporal manager processes them as soon as possible by highlighting the performance. Thus, it does not need to be done immediately, instead, the system finds the best suitable window for the processing.

Own_constraint option focuses on the user specification of the rules for removing historical data. The solution is based on a user-defined function that takes the object and time parameters as the input parameters and returns a Boolean value meaning, whether a particular object state can be removed based on the conditions or not. True value approves it, False references the rejection. If the NULL value is provided as an output, it means the temporal politics have not been set or the evaluation cannot be done due to invalid definitions or conflict constraints.

The granularity of the settings is associated with the whole database, table set, attribute, or synchronization group. The syntax of the definition is stated in the following data block:

Database level:

```
alter database
  set temporal_management =
      { static | conventional | future | full_temporal
        | restricted | timing( {Date_val | Timestamp_val} )
        | count(n) | own_constraint }
```

Table level:

```
alter table <table_name>
  set temporal_management =
      { static | conventional | future | full_temporal
        | restricted | timing( {Date_val | Timestamp_val} )
        | count(n) | own_constraint }
```

Attribute / Group level:

```
alter table <table_name>
    alter data_val <data_identifier>
  set temporal_management =
      { static | conventional | future | full_temporal
        | restricted | timing( {Date_val | Timestamp_val} )
        | count(n) | own_constraint }
```

The header of the function dealing with the own_definition is listed in the following source code:

```
Create or replace function
    temporal_constraint_mapper_func
        (id Object_ident,
         validity_frame_bd Timestamp,
         validity_frame_ed Timestamp)
  return Boolean
```

A. Data warehousing using advanced analytics

In the above section, we have been dealing with the data storage restriction using the temporal_management definition associated with the whole database, table, or attribute group. It uses the finest definition, so the table inherits the property of the whole database, the group can override the table or database selection, etc.

If the conditions are passed, particular historical states are removed. However, how to perform the Delete operation? The above solutions assume physical removal of the particular object states. In many circumstances, physical removal should be replaced by the logical operation of shifting the data from the main structure to the associated data warehouse, lake, analytical interface, or just historical repositories used as backups. One way or another, our proposed solution offers temporal registration using the external pointer. The registration is done similarly and uses either database, table, or group precision.

Database level:

```
alter database
  set historical_data_pointer = <connection_details>
```

Table level:

```
alter table <table_name>
  set historical_data_pointer = <connection_details>
```

Attribute / Group level:

```
alter table <table_name>
    alter data_val <data_identifier>
  set historical_data_pointer = <connection_details>
```

V. CONCLUSIONS

Temporal databases form an inseparable part of information technology by storing not only current valid states, but the states valid in the past, as well as future perspectives can be covered. In terms of temporality, multiple states are stored for one object, enclosed by the temporal sphere frame, mostly representing the validity.

Through this paper, various temporal architectures were discussed, highlighting the object, attribute, and synchronization group granularity. The core architecture focuses on the current state separation. Thanks to that, existing conventional applications do not need to rebuild their solutions, whereas the conventional data structures remain original.

The main contribution of this paper is hybrid temporal architecture, which is group granularity oriented but highlights any temporal sphere management. Thanks to the provided solution, conventional, temporal, and static attributes can be covered, and delimited by the temporal registration applied for the whole database, table, synchronization group, or attribute itself. Any potential collision in terms of temporal management is solved by finer precision, which can override the original selection.

Historical data do not need to be stored in an unlimited manner permanently. Therefore, various enhancements and parameters were proposed, focusing on the physical removal or transfer of such states into the data warehouse, data lake, or analytical interface, always registered and managed by the temporal layer.

By using these techniques, methods, and models, complex architecture can be developed, focusing on the temporal data efficiency, delimited by the error-prone, reliability, accessibility, etc.

Although the proposed techniques are delimited by the temporality frame, the same solution can be applied to the spatio-temporal models, as well.

In recent future research, our emphasis will be on the temporal data distribution techniques by grouping object states physically to the same data structures by using partitioning techniques. Namely, in the temporal systems, the state evolution of one object is often monitored and evaluated, thus the system should optimize the data access for the object timeline orderability.

Another research perspective will be related to the undefined values or partial undefined states by using NULL values, which cannot be commonly categorized.

ACKNOWLEDGMENT

This publication was realized with support of Operational Program Integrated Infrastructure 2014 - 2020 of the project: Intelligent operating and processing systems for UAVs, code ITMS 313011V422, co-financed by the European Regional Development Fund.

EUROPEAN UNION
European Regional Development Fund
OP Integrated Infrastructure 2014 – 2020

MINISTRY
OF TRANSPORT
AND CONSTRUCTION
OF THE SLOVAK REPUBLIC

It was partially supported by the Erasmus+ projects:

- Project number: 2022-1-SK01-KA220-HED-000089149, Project title: Including EVERyone in GREEN Data Analysis.
- Project number: 2020-1-HR01-KA226-HE-094713, Project title: Cloud cOmputing for Digital Education Innovation.
- Project number: 2021-1-SI01-KA220-HED-000032218, Project title: Better Employability for Everyone with APEX.

REFERENCES

[1] D. Kuhn and T. Kyte, Oracle Database Transactions and Locking Revealed. Oracle Database Transactions and Locking Revealed. (2021). https://doi.org/10.1007/978-1-4842-6425-6

[2] D. Kuhn and T. Kyte, Expert Oracle Database Architecture: Techniques and Solutions for High Performance and Producitivity. 2021, ISBN – 9781484274989

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

[4] D. Kuhn, S. Alapati, B. Padfield, “Expert Oracle Indexing Access Paths”, Apress, 2016.

[5] M. Kvet, “Managing, locating and evaluating undefined values in relational databases”. 2020

[6] 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.

[7] M. Kvet, K. Matiaško, M. Kvet, “Complex time management in databases”, In Central European Journal of Computer Science, Volume 4, Issue 4, 2014, pp. 269-284.

[8] M. Kvet and M. Vajsová, “Transaction Management in Fully Temporal System”, 2014. UkSim, Pisa, pp. 147-152.