# Autonomous Temporal Transaction Database

Michal Kvet
University of Žilina
Žilina, Slovakia
Michal.Kvet@fri.uniza.sk

***Abstract*—The data amount to be handled, managed, and evaluated is enormous. Intelligent information systems need to cover not just conventional data types. The whole evolution should be addressed instead. Cloud environment offers a wide range of opportunities pointing to the scalability of the solution. Oracle Cloud databases are based on the autonomous processing of the system administration done by machine learning and artificial intelligence. Autonomous database processing core element states mainly just the transaction control of the current valid states. In this paper, we deal with the temporal architectures pointing to the object, attribute and synchronization group granularity. The internal representation must be currently defined by the user explicitly. Our proposed solution is fully autonomous, covered by the Oracle Cloud technology. User just register the structure to be managed temporaly, granularity is maintained by the system, following the complexity and scalability of the solution. Data management is extended by the Granularity strategy management module to ensure either storage efficiency, as well as the format of the query result set. Introduced DBMS_AUT_TEMPORALITY package registers the data to the temporal sphere and limits the historical data reflecting the information value.**

## I. Introduction

Information technology can now be found almost everywhere. Each of us needs data for proper decision-making. Database technology is a core, robust, and performance-effective element for dealing with the data by delimiting the data and operations themselves. Transaction control is now the most widespread system used in intelligent information systems. It can be clearly stated that database technology forms an inevitable part of each information system. Data to be handled are made more and more complex. It is not even suitable to deal just with the current valid states [1], [2]. The temporal evolution is the whole time spectrum. Data management is inevitable for the complexity and robustness of the decision making, creating prognoses, data analytics, trends identification, etc. Currently, most databases are shifted to the cloud environment highlighting the general approach, anytime availability, security, and reliability [4], [5]. The performance of the system is an important aspect, as well. Cloud systems that are flexible can automatically reflect the workload by scaling the sources, storage, memory, and CPU.

Moreover, data reflection, security, and backup strategy are maintained automatically without the user intervention necessity. Database administrators have then different roles. Internal database administration is shifted to the cloud itself.

In this paper, the Oracle database system and its cloud environment are used. The selection was made based on the project *CodeIn* (*Cloud computing for digital education innovation*). This paper forms one of the outputs of such a project implemented based on the Oracle Corporation partnership.

This paper summarizes the autonomous database technology, pointing primarily to the transaction processing handling conventional systems and the relational system evolution summary. The main contribution of the paper is delimited by the temporal extension of the autonomous database processing by defining dynamic data structure reflecting the temporal spheres. Spatial dimensions can be covered, as well. New autonomous temporal architecture is introduced. It is based on the conventional data model passed to the system and the temporal monitoring. The internal structures, management, and evaluation it then automated, query results are formatted based on the user requirements and are not influenced by the storage format. Thank to the proposed solution, performance, robustness, and scalability can be ensured.

The paper is organized as follows. Section 2 deals with the Oracle relational database from the on-premise world to the cloud environment. It points to autonomous database processing, as well. Section 3 points to temporal structures by proposing its own technique of autonomous temporality management inside the Oracle Cloud in section 4. Finally, computational study and performance evaluation are stated in section 5, highlighting the data retrieval process – the processing time and storage repository demands.

## II. History of Oracle reflecting relational database technology

In the first phases, Oracle was initially called Software Development Laboratories, pointing primarily to the consultation services in the area of programming. Their first commercial product was just a database system called CODASYL. It was based on the relational algebra introduced by Edgar Codd in the 70ties of the 20$^{th}$ century allowing the creation of flexible and independent data management architecture to store, manage, evaluate, aggregate, and access the data by the queries providing the results in the desired format and quality. Thus, the data themselves were physically and logically separated from the application itself. Before that, such data were directly embedded. It allows systems and developers to interconnect data and access them parallelly from various applications and systems. Edgar Codd was employed by the competing company IBM developing a database management system for the mainframe computer [1], [2], [6], [7].

ISSN 2305-7254

Initially, Oracle founders Larry Ellison, Bob Miner, and Ed Oates were full of worries about usability and results. However, they agreed to create their system using the mini-computer architecture instead of a mainframe. Such an approach was soon hired by well-known companies like NSA, CIA, or Navy intelligence resulting in building still more and more complex systems. The original company name was changed to the Oracle Corporation during this successful period, focusing primarily on the databases. Besides, they cooperated on multiple projects like Ingres (Interactive Graphics Retrieval System). They created their system language called QUEL. However, the ANSI standard accepted the IBM version – Serial Query Language [2], [10].

Over the years, the Oracle database was implemented, improved, and extended, focusing on the technologies at the time, namely internet, grid, and cloud. Version 12c, introduced in March 2017, strongly focused on cloud technology by shifting the on-premise world to the autonomous cloud environment providing robust, reliable, and performance effective architecture on its Exadata servers optimized for the database management (composed by the Sun Microsystem acquisition in 2010) [20].

The headline of the cloud data management and transformation was based on the fact that "*the cloud is just someone else's computer"* resulting in *creating Oracle Cloud Infrastructure* (*OCI*). In 2010, several architecture enhancements were provided, like *Infrastructure as a Service* (*IaaS*) followed by consecutive changes and opportunities in *Software as a Service* (*SaaS*) or *Platform as a Service* (*PaaS*) delimited by the available sources and methodology used inside [14], [15], [18].

The complexity of the cloud was offered at the end of 2016, focusing on virtual machines, networking, and containers [19].

A significant innovation is just the *Autonomous Database Processing* supervised by the transaction support, ensuring data reliability, integrity, and consistency.

## III. Temporal structures & autonomous management

*Autonomous Transaction Processing* (*ATP*) is an Oracle Cloud service ensuring performance and robustness by eliminating the operation management complexity. It automates provisioning, configuring, tuning, and scaling the database management, so the user does not need to manage and administer it explicitly. Moreover, it is ensured by the auto-repairing and patches applied automatically. From the optimization point of view, it is organized in a multitenant, distributed, fragmented, and partitioned environment. Due to the advanced compression option and the hardware capabilities, data can be accessed very effectively, even with the emphasis on the memory data location by using in-memory type. Sophisticated optimized system from the internal hardware up to the software and availability allows complex data management across the time spectrum and dimensions [1], [10], [11].

Provided autonomous operations are [20]:

- *Auto-provisioning* reflecting fault tolerance, availability through any location anytime.
- *Auto-tuning* – automated structure, storage, and memory optimization.
- *Backup strategy* – automated backup strategy performing full backups weekly and incremental backups launched daily to ensure the possibility to restore the database to any timepoint up to the past 60 days.
- *Auto-repairing* – failure prediction and data duplications across the availability domains.
- *Auto-failover* – any downtime elimination by using *Autonomous Data Guard [3], [12]* architecture switching to the remote copy automatically.

The main point of the *ATP* is related to the transaction processing covering mostly current valid states. Historical data can be partially found in *transaction logs,* ensuring the data operation *consistency* (data operation must process the data valid at a point of the start, irrespective of any change applied during the process itself). As stated, 60-day history can be found in the backups, either full or incremental. As a consequence, previous data states and existing tuples can be identified during such a period. However, it is a complicated process resulting in significant time consumption and resource demands. Fig. 1 shows the UML diagram of the data flow. First, the whole restoration point is identified, which can be reflected mainly by the full backup. However, generally, the current database image can be used, as well. Then, historical data image is obtained by using daily incremental backup focusing on the execution time. Thus, archived and online logs can be inevitable to apply if no newer daily backup is present. As evident, two problems can be identified – data do not need to be covered by the backup – firstly, only 60-day historical data are generally available. Secondly, if the database is not in an archive mode, some states can be missing, not covering the whole transaction set. It is evident that the daily granularity is covered. However, it cannot be directly divided into individual transactions.

As a result, although the availability is ensured, it is still limited in terms of the history and the performance and demands. As evident, the whole backup has to be loaded, followed by additional structure processing. Future valid data cannot be managed at all. *OCI ATP* can provide scalability and availability for robust data structures and large data sets. Thanks to the configuration, the whole time spectrum can be used. In the following part of the section, a summarization of the temporal architectures is present, followed by the own proposed solution for autonomous temporal data management inside the system. Thanks to that, any time granularity can be used.

Moreover, individual attributes and tuples can be extended by the temporal accessibility and monitoring principles, which are then evaluated without any user intervention. On the other hand, it is just the storage perspective optimization. The result set is always composed as required, irrespective of the internal representation.

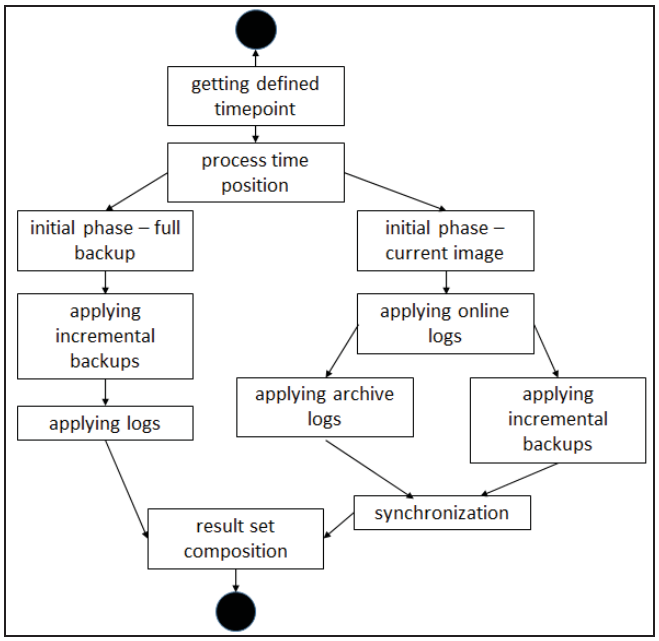



Fig. 1. Process of obtaining historical data

### *3.1 Temporal architectures*

Several architectures and enhancements can be identified over the decades. However, they are primarily based on the conventional architecture extension instead of using own temporal paradigm. It is caused by the fact that the ANSI standardization process was not done and accepted. Thus, individual database systems were forced to develop their own systems or leave the definition covered by the end-user.

The conventional database approach is characterized by storing only the current valid state. Any change delimits the physical update by replacing the original value with the newer one. Thus the history cannot be complexly found at all. By introducing transaction logs supervised by the archiving process delimited by copying online redo logs to the archive repository before the physical rewritten, historical data can be obtained by extracting transaction data from such logs. Although it was technically possible, it was too demanding, whereas the transaction header does not point to the manipulated objects, so the operations were needed to be extracted and evaluated step by step sequentially. In a more complex system, such processing was infeasible in terms of time demands. Even the shift to the cloud environment would not reach any relevant solution.

The more sophisticated temporal architectures were consecutively defined by the object's primary key extension. Thus, the state itself was not identified by the object definition. However, the time frame has to be specified, as well. The uni-temporal data model was based on one temporal dimension (primarily delimited by the validity). However, bi-temporal and multi-temporal dimensions can generally be used to reflect the reliability, transaction time frame, synchronization timestamps, etc. [16], [17].

The above-defined temporal architecture extends the primary key definition by the time elements forming object-level temporal architecture. The time frame delimits each object state, so any data change requires a new whole state to be defined. Non-changed values cannot be delimited by the NULL notation, whereas such values can have a specific representation. A partial solution can be identified by pointing undefined values to specific memory structures delimited by the causality.

An attribute-oriented temporal system shifts the granularity to the attribute itself. The state is then composed of individual attribute sets delimited by the time representation definition or any time element specified in the system, respectively. The architecture of the attribute-oriented granularity is shown in fig. 2. It consists of three layers. Current states are in the first layer, historical as well as valid, future states are separated into the third layer, which is also attribute oriented. The core part of the system is formed by the temporal layer locating and storing the pointer to any change of the temporal attribute in time. Thus, any change, as well as the whole attribute state evolution, can be identified and is accessible [16], [17].

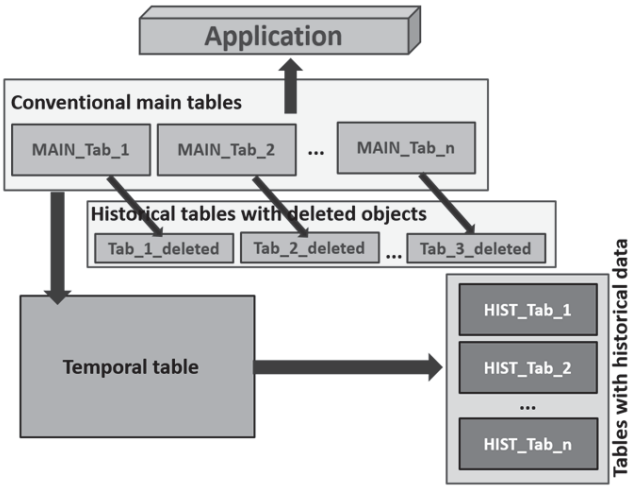



Fig. 2. Attribute oriented granularity [16]

Data flow of the Update operation consists of the inner granularity detection, real change identification preceded by state forming. If a new object is added to the system, principles are similar. Delete operation can be performed either logically by marking the object or by the physical removal operation. In that case, two options are available – the state is completely removed from any layer or the evidence of the state's existence is still present, but the values are hidden.

A general solution is provided by the group granularity, by which the data updates can be synchronized in the temporal layer. In comparison with attribute granularity, where each attribute change forces the system to load a new data row into the temporal layer, group granularity detects synchronization

groups, which are then maintained as just one attribute. Naturally, all such data are temporally oriented. Fig. 3 shows the architecture.

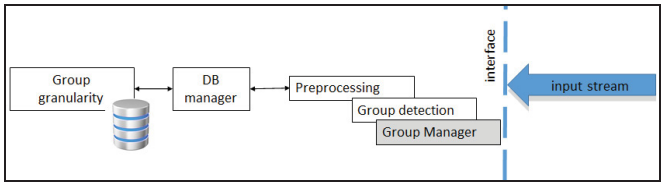



Fig. 3. Group granularity

The above architectures are based on the temporality reflected by the various granularity levels. Although the management can be partially automated by machine learning techniques to identify synchronization groups, the existing solution cannot be categorized as autonomous in terms of definition and performance. Section 3.2 deals with the own proposed solution based on the attribute association to the temporality module expressing the temporal sphere to be handled and processed. The solution is entirely autonomous in terms of the architecture, individual system sources, up to the whole management, and result set providing. The aim is to provide the data in a requested form irrespective of the internal data model and time representation.

*3.2 Proposed solution – autonomous temporal management*

The main limitation of the explicit temporal data definition is based on the user intervention necessity by adapting the temporal requirement to the data model and triggering the change by passing the evidence to the temporal layer. If the synchronization data management is present, the problem is even deeper, whereas, for each changed attribute, the particular trigger is fired. However, it is necessary to evaluate the group, not separate attribute set. As a result, management is complicated. Dynamic synchronization group handler must be applied either to the temporal layer but to the triggering events and data dictionary, as well. If the detection and group reconstruction is dynamic, the system uses too many resources to optimize the internal structure, with no reflection to the input data themselves. Our proposed solution combines all these three temporal architectures (object, attribute, group granularity) to the one common ***autonomous temporal solution*** (*ATS*). The overview of the architecture and principles are shown in fig. 4.

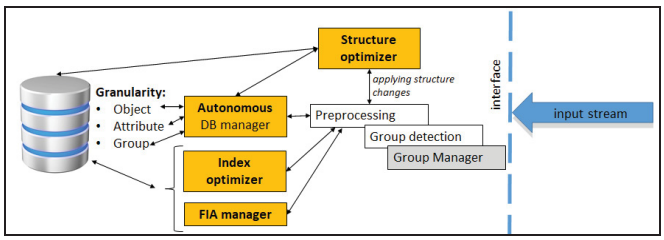



Fig. 4. Autonomous temporal solution architecture

First of all, the data model is provided to the system, or the data source connection can be provided by reverse engineering extracting the data structures from the source. As a result, the conventional data model is reflected in the first step. Then, individual tables and attributes are covered by the temporal reflection. For each attribute or the whole table, temporal spheres are defined. The validity mostly delimits them. However, as already stated, the multi-temporal solution can be determined, as well, by prompting the system to cover other temporal spheres, like transaction validity, insertion time, synchronization time in offline systems, etc. Afterward, the system is autonomous. Internal structures are created reflecting the data pre-processing and dynamically adjusted based on the input data flow. As a result, storage demands are optimized, the temporal layer is not so overloaded. Fig. 5 shows the creation management process.

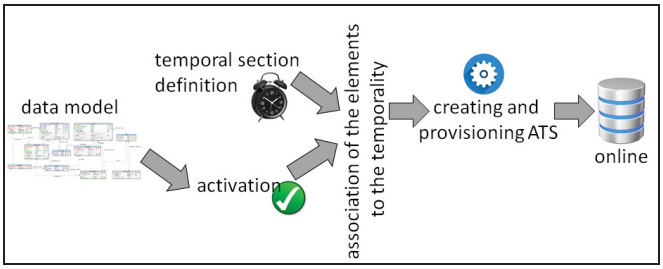



Fig. 5. ATS definition and provisioning

Fig. 6 shows the staged process of the data retrieval. Note that the result set structure is *Select statement oriented* and does not rely on the internal data representation at all. For the optimization, whereas several systems can be connected just to the conventional layer, current valid states are always accessible and pre-prepared in a logical layer.

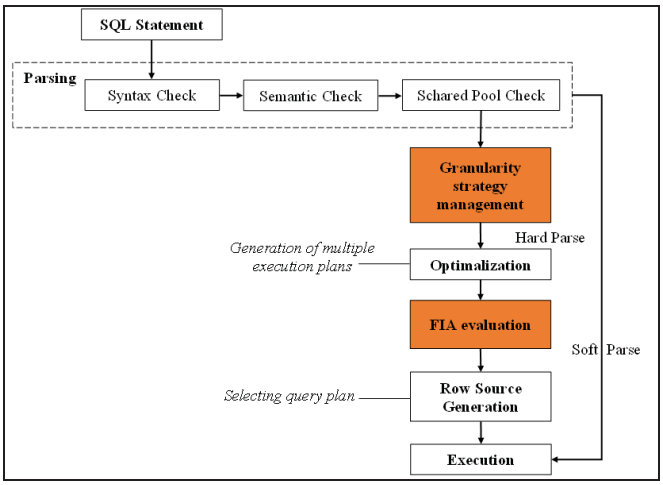



Fig. 6. Data retrieval process

Fig. 6 shows the process of data identification physically in the database. Data can be accessed either by sequential scanning of the data blocks or by using an index [13]. All these activities are done automated, as well, to ensure global performance. Index set is created and altered dynamically, based on the workload, queries, and partitioning. Data are automatically partitioned. Each partition fragment has its own index set. Thanks to that, global performance is ensured, as visible in the following section.

Moreover, *Flower Index Approach (FIA)* introduced by our team in [17] is used, as well. Its principle is based on data identification and location in a fragmented block environment. In a real temporal system, data block fragmentation is hugely present. If no suitable index is present in the system, the optimizer selects the sequential scanning method of the all data block set associated with the table. Such an option has, however, many negative aspects. Some blocks can be even empty, fragmented or the migrated rows can be present [8], [9], [15]. Thanks to that, performance can significantly degrade, whereas each block must be database loaded to the memory buffer cache for the evaluation. Although the cloud systems use robust hardware and disc interfaces, significant delays caused by the system source consumption can be detected. As a result, we have introduced the *FIA* approach, which is based on the relevant *data block identification* by reducing the standard *B+tree index* just to the block reference – *BLOCKID*.

In comparison with original sequential scanning, empty blocks are removed from the evaluation. Thanks to that, indexes do not need to be rebuilt, which is so demanding operation. In a dynamically changing temporal environment, consumption is even more shaped.

Data flow management is done by defining the data model in the first phase, followed by the temporal ranges specification. It is done by using *Create temporal section* command, which delimits the time definition to be handled. It can represent any temporal sphere, reflected mainly by the validity. Each temporal section is uniquely identifiable by its name:

```
Create temporal section temporal_section_name
  identified as [validity | transaction_insert | transaction_commit
                | synchronization_time | processing_time
                | distribution_time ...]
    by [timepoint | time_interval]
    using precision [date | timestamp[(n) | date_elements(element_list)];
```

To register the temporal dimension monitoring, *REGISTER* procedure of the proposed *DBMS_AUT_TEMPORALITY* can be used consisting of three parameters – name of the table (optionally prefixed by the object owner), name of the attribute, and created temporal section identified by its name:

```
DBMS_AUT_TEMPORALITY.REGISTER(table_name varchar,
                              attribute_name varchar,
                              temporal_section varchar);
```

The name of the attribute is optional and can be omitted. In such a case, the registration is done for the whole table. Similarly, the opposite operation of the de-registration can be used, as well.

```
DBMS_AUT_TEMPORALITY.DEREGISTER(table_name varchar,
                                attribute_name varchar,
                                temporal_section varchar);
```

If the table or attribute specification is not done, it is generally treated as a conventional table. To override it, the definition can be done on a system level, which is then used as a default option:

```
DBMS_AUT_TEMPORALITY.SET_DEFAULT
                              (temporal_section varchar);
```

In principle, monitoring of the data value management and evolution allows unlimited access to the history. Plans can be covered, as well. To limit the amount of the data, mainly in a historical manner, the following methods can be used. They are encapsulated in the DBMS_AUT_TEMPORALITY package so that the overloading can be used. The first code deals with the total amount of historical state limitations. The second code specifies the minimum duration, which must be passed to remove the state. It is expressed in seconds. Similar to already described principles, it can be set for the whole table or individual attributes. In the case of using only attribute granularity, the historical data state can be only partially valid. Some attribute values would be restricted.

```
DBMS_AUT_TEMPORALITY.RESTRICT_ACCESS
   (table_name varchar, attribute_name varchar, history_count integer);
DBMS_AUT_TEMPORALITY.RESTRICT_ACCESS
   (table_name varchar, attribute_name varchar, duration integer);
```

Finally, such an approach can be associated with specific users or can be used generally. Thanks to that, each user (or role) can have specific data access, monitoring the temporality, as well as access to the whole history. Historical data are removed by planning the automated jobs at a defined time point or as a result of a new state definition.

## IV. COMPUTATIONAL STUDY

Performance evaluation has been performed using the *Oracle Cloud* environment located in the *Frankfurt* data region – *Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 – Production Version 21.2.0.0.0*. The storage capacity was *20 GB* for internal data management. Backups were part of the *Object storage* outside the database itself. The database consisted of the spatio-temporal data locating and identifying airplane objects by the occurrence time, GPS position, as well as other parameters – speed, destination, current airspace association (entry and exit time), planned route vs. current route, as well as the weather conditions influencing the flight itself. The following categorization covered 100 attributes:

- 20 attributes were static with no evolution over the time characterizing the airplane and airport properties,
- 20 attributes were covered by the entities monitored using object-level temporal architecture.
- 30 attributes were separately monitored using attribute granularity.
- 30 attributes can be used for group detection and synchronization, whereas all data were updated at the same time. These attributes were part of 5 tables consisting of 6 attributes.

```
"ECTRL ID","Sequence Number","AUA ID","Entry Time","Exit Time"
"186858226","1","EGGXOCA","01-06-2015 04:55:00","01-06-2015 05:57:51"
"186858226","2","EISNCTA","01-06-2015 05:57:51","01-06-2015 06:28:00"
"186858226","3","EGTTCTA","01-06-2015 06:28:00","01-06-2015 07:00:44"
"186858226","4","EGTTTCTA","01-06-2015 07:00:44","01-06-2015 07:11:45"
"186858226","5","EGTTICTA","01-06-2015 07:11:45","01-06-2015 07:15:55"
"186858227","1","EGGXOCA","01-06-2015 04:08:00","01-06-2015 05:01:00"
"186858227","2","EISNCTA","01-06-2015 05:01:00","01-06-2015 05:34:00"
"186858227","3","EGPXCTA","01-06-2015 05:34:00","01-06-2015 06:18:10"
```

Fig. 7. Input data example

The amount of data evolved over time. It used the principle of dynamic allocation of the 20 GB distributed to individual states. If the limit was reached, historical data were transferred to the data warehouse.

For the computational study, five architectures were used. All of them used autonomous transaction processing databases. However, the temporal management was different. Namely, the first model (*MODEL 1*) deals just with object-level temporal granularity. The second model (*MODEL 2*) is based on just the attribute granularity irrespective of the synchronization itself. *MODEL 3* uses group granularity for processing. Synchronization groups are detected automatically. Thus the temporal layer storage is optimized. *MODEL 4* is based on the proposed solution covered by this paper. It is based on autonomous processing. Data management is treated entirely automatically. The index set was managed automatically. However, *Flower Index Approach (FIA)* was not used, all. Finally, *MODEL 5* uses the same principles as *MODEL 4*, but the *FIA* approach locating relevant data blocks in case the non-suitable index for the query is present. In the first evaluation criterion, the Always Free option of the Oracle Cloud was used, with no scalability option. Fig. 8 shows the results – the data retrieval process forming the shape of the whole object state.

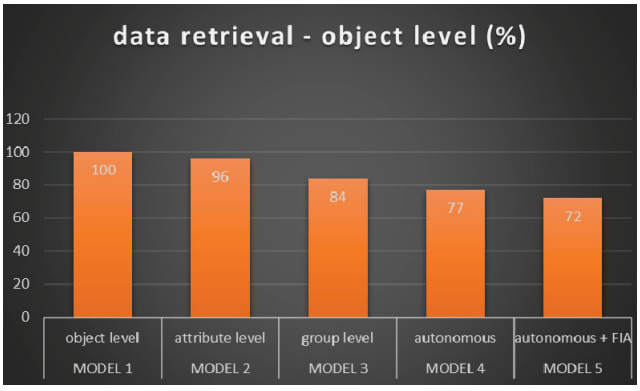



Fig. 8. Results – data retrieval using object level output format (%)

All the data values are expressed in percentage for better visibility and evaluation.

The first part of the computational study is based on the process of data retrieval. The output consists of 10% of the whole data set projecting one table. The requested format is object level, regardless of the real change identification (object-level temporal architecture forms a new state irrespective of the real change consequencing in the necessity to store several values multiple times). Object-level temporal architecture is a referential model, so the total demands are 100%. Although it is evident that the data shape is optimal for the output, attribute level granularity even reaches a 4% improvement. It is caused by the data amount reduction to be memory-loaded. Although the whole state should be composed of the individual attributes, the total number of blocks to be processed is significantly smaller.

Moreover, using just attribute granularity lowers the impact of data fragmentation. Synchronization group management is more helpful in decreasing the processing time demands up to 16%. It is caused by reducing the data pointer amount in the temporal layer, which is responsible for the state calculation and reflection. By applying autonomous management, all three categories can be present, optimized dynamically based on the data inputs and structure. Although the group level temporality is optimal from the evaluation point of view, dynamic group detection has additional demands on the storage and consecutive loading, whereas the group properties must be stored to be able to reproduce the change.

Moreover, such groups are dynamic evolving. As a result, the total processing time demands are lowered to the value of 77%. Finally, by applying dynamic index set management supervised by the FIA approach, data access can be more reliable, trusted but mostly faster. Namely, the total processing time requirements compared to the object level architecture are lowered up to 23%. Comparing *MODEL 4* and *MODEL 5*, demands are reduced using 6,5% by removing any impact of the fragmentation.

Fig. 9 shows the results, which were obtained if the result set focuses just on the real changes during the time frame – newer value must be different from the already existing direct predecessor. Otherwise, the value is ignored. Each object state provided by the object level architecture must be compared to the previous state forcing the system to sort the states and categorize the change. Such demands are increased from the value 100% to 130%, comparing object-level results and real change identification. Attribute architecture is more suitable, whereas only real change is stored, secured by the pre-processing during the loading process. The total demands are then 92%. Group level granularity optimizes the temporal layer by reducing the processing to 81%.

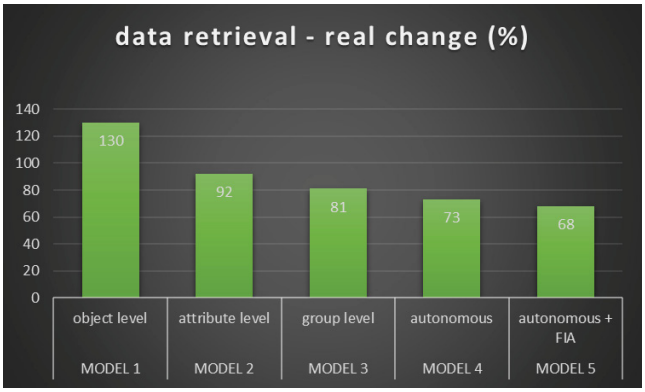



Fig. 9. Results – data retrieval using real change identification (%)

In comparison with attribute granularity (81%), it reflects the 9,9% improvement. Finally, autonomous processing uses all strategies, so the demands are 73% with conventional processing and 68% with complex extension. Thus, the FIA approach reduced the demands from 73% to 68% reflecting the 6,8% improvement.

Fig. 10 shows the storage demands across the used architectures. Reached results are part of the figure in a graphical style. The reference model is object-level temporal architecture. 20% of data are static, 20% are object-oriented. In a theoretical manner, the reduction can reach 40% for the attribute perspective. The real performance evaluation showed the 34% improvement caused by the additional structures located in the temporal layer. Group level minimizes the storage demands more strictly, reaching the 41% improvement. Autonomous processing requires only 46%. Each attribute or table is separately temporaly treated by excluding the temporal layer processing for the static and object-level temporality. Thanks to that, the total storage demands can be rapidly lowered. As you can see, the *FIA* approach requires 1% for storing dynamic block indexes (comparing object level and autonomous *FIA*).

Comparing autonomous temporal data processing (*MODEL 4*) and FIA extension (*MODEL 5*), the additional disc capacity requirement is 2,2%.

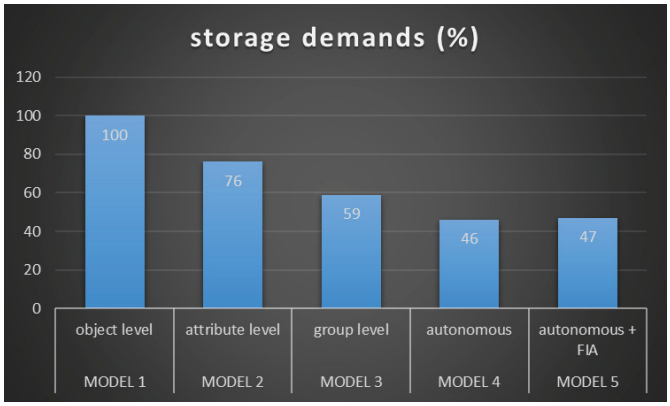



Fig. 10. Results – storage demands (%)

In the second part, the solution's scalability is highlighted to extend the data set size up to 50, 100, and 200GB. Fig. 11 and Fig. 12 show the results emphasizing the data output structure – object level or real change identification. As evident, there is no significant change if the data amount is enormously extended. With the data accumulation, group and autonomous approaches benefit due to the broader synchronization management possibility.

Similarly experimented, the total OCPU amount availability extension distributes the workload proportionally.

## V. Conclusions

Data amount to be handled, operated, stored, and retrieved is continuously rising. It is evident that the conventional principles keeping just current valid states are not suitable in intelligent information technology anymore. For the decision making, analytics, prognosis creation, complex data should be present covering historical, current data, as well as plans. The relational database system is covered by relational algebra and transaction support. This paper deals with the temporal architecture overview. The main contribution is related to the own proposed autonomous temporal database extending the autonomous transaction processing option (ATP) available through the Oracle Cloud technology. The proposed solution is fully temporal and allows user to let the system manage your data complexly in a spatio-temporal environment. By using the proposed technology, data retrieval demands can be significantly lowered. Namely, based on the computational study, namely 19,8% for the attribute system and 8,3% for the group granularity.

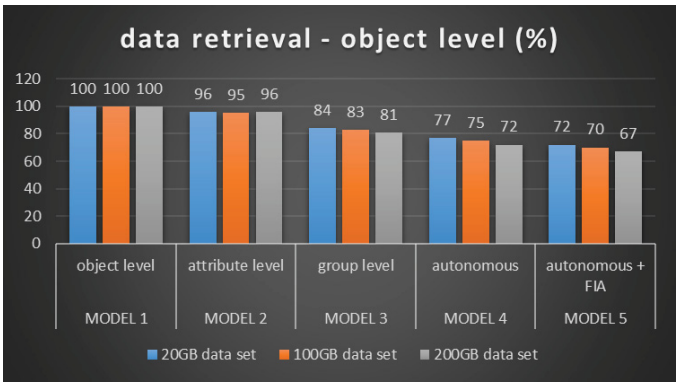



Fig. 11. Scalability handling – object change identification (%)

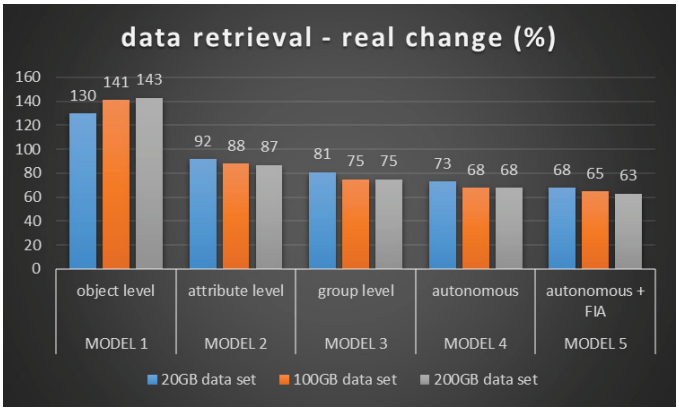



Fig. 12. Scalability handling – real change identification (%)

New composition of the autonomous temporal environment registration has been proposed by creating additional processes supervising the architecture. First of all, the original data model is passed to the system, followed by the tables and attributes extraction. For each element, temporal monitoring can be registered. Afterward, the whole structure, inner management, and storage optimization are done autonomously. The result set of the query is then automatically formatted based on the user criteria, regardless of the internal representation. Thus, storage and access processes are optimized.

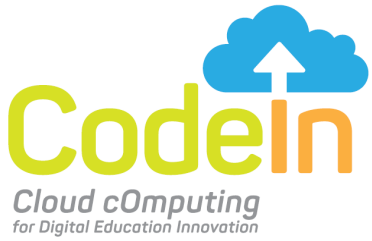
In the future, we will highlight the data distribution in multiple availability regions and sharing the data across the tenancies.

## ACKNOWLEDGMENT

This publication was realized with the support of the Erasmus+ project:

Project number: 2020-1-HR01-KA226-HE-094713

Project title: Cloud cOmputing for Digital Education Innovation




## REFERENCES

[1] Abhinivesh, A., Mahajan, N.: The Cloud DBA-Oracle, Apress, 2017

[2] Anders, L.: Cloud computing basics, Apress, 2021

[3] Amirishett, A., Li, Y., et al.: Improving Predictable Shared-Disk Clusters Performance for Database Clouds, 33rd IEEE International Conference on Data Engineering (ICDE), 19-22 April 2017

[4] Castro-Leon, E., Harmon, R.: Cloud as a Service, Apress, 2016

[5] Elbahri, F., Al-Sanjary, O., et al.: Difference Comparison of SAP, Oracle, and Microsoft Solutions Based on Cloud ERP Systems: A Review, 15th IEEE International Colloquium on Signal Processing & Its Applications (CSPA), 8-9 March 2019

[6] Jakóbczyk, M.: Practical Oracle Cloud Infrastructure: Infrastructure as a Service, Autonomous Database, Managed Kubernetes, and Serverless, Apress, 2020

[7] Meier, A., Kaufmann, M.: SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management, Springer, 2019

[8] Mikkilineni, R., Morana, G., Keshan, S.: Demonstration of a New Computing Model to Manage a Distributed Application and Its Resources Using Turing Oracle Design, 25th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 13-15 June 2016

[9] Moussa, R.: DDB Expert: A Recommender for Distributed Databases Design, 22nd International Workshop on Database and Expert Systems Applications, 22 August - 2 September 2011

[10] Pendse, S., Krishnaswamy, V., et al.: Oracle Database In-Memory on Active Data Guard: Real-time Analytics on a Standby Database, 2020 IEEE 36th International Conference on Data Engineering (ICDE), 20-24 April 2020

[11] Png, A., Demanche, L.: Create Modern Web Applications Using Always Free Resource, Apress, 2020

[12] Tanveer, A.: Oracle 19c Data Guard, 2020

[13] Kriegel, H., Kunath, P., et al.: Acceleration of relational index structures based on statistics, 15th International Conference on Scientific and Statistical Database Management, 9-11 July 2003

[14] Kumar, Y., Basha, N., et al.: Oracle High Availability, Disaster Recovery, and Cloud Services: Explore RAC, Data Guard, and Cloud Technology, Apress, 2019

[15] Kumar, A.: Oracle 12c Data Guard Administration: Learn to protect your database against planned & unplanned downtimes, Independently published, 2020

[16] Kvet, M, Matiasko, K., Kvet, M.: Complex time management in databases, Central European Journal of Computer Science vol.4, 2014, pp. 269-284, doi: 10.2479/s13537-014-0207-4

[17] Kvet, M.: Managing, locating and evaluating undefined values in relational databases. 2020

[18] Riaz, A.: Cloud Computing Using Oracle Application Express, Apress, 2019

[19] Wang, W., Tian, N., et al.: Testing Cloud Applications under Cloud-Uncertainty Performance Effects, 11th IEEE International Conference on Software Testing, Verification and Validation (ICST), 9-13 April 2018

[20] Oracle Cloud Free Tier, https://www.oracle.com/cloud/free