https://doi.org/10.58245/ipsi.tir.2502.03

Building Temporal Architecture on Transaction Logs

Kvet, Michal

Abstract: ***Conventional databases are related to the relational paradigm and store current valid data. This means that any change in operation replaces the original state. History cannot be directly obtained. However, transaction logs and internal change vectors can offer the ability to construct states as they existed in the past. Flashback technology introduced in the Oracle database is primarily intended for security, reliability, consistency, and recovery. The process of obtaining historical images can be strongly resource-demanding. Our proposed solution enhances transaction logs by indexing them, delimited by the data pointers. Thanks to that, processing time and costs can be significantly reduced. Users deal only with the conventional layer, and historical data is automatically available through effective log handling.***

Index Terms: ***conventional paradigm, relational data, transaction logs, indexing, performance***

1. INTRODUCTION

CONVENTIONAL databases are systems that store and manage data in a structured way and formats in database schemas. They are commonly operating large amounts of data. Given the huge amount of data being generated, it is very important to ensure the efficiency of processing and data access to ensure quick data retrieval, updates and overall manipulation of the stored data. These are key features of the conventional databases [8] [9]:

• *The relational data model* organizes data in the form of tables and relationships between them, handled by the data model and core constraints, like data type, NULL, unique, etc.

• *SQL language* that serves data operations and manipulation. It is used for querying, inserting, updating, deleting, and, respectively, moving data to other repositories. Regarding data change, it is important to mention referential integrity to make the database consistent after the operation (transaction) itself.

• *Transaction support* ensuring data integrity, reliability, consistency and value prone, delimited by the ACID properties – atomicity, consistency, isolation and durability. A core element related to transaction management is just the integrity delimited by the primary keys, foreign keys, domains, columns and additional user-defined constraints.

• *Performance* – since the structure is precisely defined, there is a wide range of possibilities for ensuring performance, from indexing through architecture to concepts of data distribution and in-memory column storage.

Currently, there are several types of architectures, which can be covered by the conventional paradigm:

• *Relational databases*, like Oracle, MySQL, PostgreSQL or Microsoft SQL Server

• *Hierarchical databases*, like IMS - Information Management System by IBM

• *Network databases*, like IDS – Integrated Data Store.

The main advantages of those systems include consistency, scalability, integrity and supporting tool maturity. Vice versa, limitations are rigid schema, and low efficiency for non-structured data, like multimedia, document files or huge logs. Among that, as the name suggests, conventional databases deal only with current valid data. It means that any Update operation physically replaces the original state. Thus, after the operation, there is no evidence of the original state in the system. There is no evidence about the number of previous Updates, as well. Furthermore, if an object is deleted, information about its existence is lost. Based on [1] [8], historical data of the conventional system can be partially found in transaction logs. Each database transaction generates change vectors, allowing historical data to be processed using them. Principles, approaches and limitations are discussed in the first part of section 2. The main limitation of that management is related to the accessibility and availability over a long period. However, the most important fact to mention is that such a system cannot meaningfully process the states of objects in time to create a temporal system. States that are valid in the future are not covered at all.

To make historical data available in an effective way, it is necessary to reliably locate relevant data. Our proposed solution described in section 3 is based on complex indexing of the online and archive logs, enhanced by the post-indexing techniques.

It can be even applicable if the archive

transaction logs are not available through its extraction module storing filtered logs in the newly introduced archive repository.

Performance evaluation study of various concepts and solutions is covered in section 4.

This paper aims to create temporal database enhancements on the architecture level but also focuses on the index layer, making the data access location smooth and efficient. The whole system is based on data analytics but can also be connected to the transaction-oriented system. Fig. 1 shows the core principle of the system's applicability. Transaction-oriented system (OLTP) is conventional, temporal data are delimited by the analytical database (OLAP), on which the proposed architecture is composed. Yellow arrows represent data loading, and red arrows express data retrieval processing.

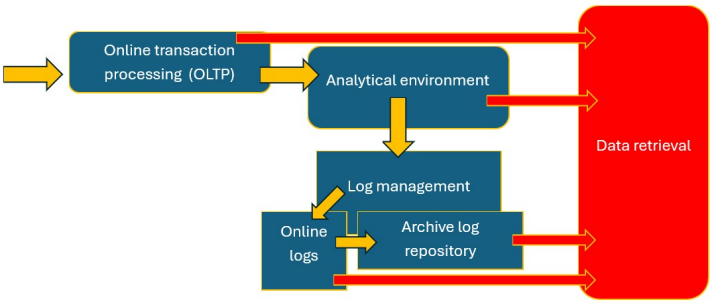

Figure 1. Conventional data and log management

The proposed solution is primarily used for environmental data analysis as the output of the VEGA 1/092/24 project - Developing and applying advanced techniques for efficient processing of large-scale data in the intelligent transport systems environment. Inspiration and data were provided by the implementation of the Erasmus+ project Evergreen – Including Everyone in Green Data Analysis [19].

Environmental data analysis involves the process of collecting, interpreting, and using data related to environmental factors, such as air quality, water quality, climate, biodiversity, and more. The goal is to understand the state of the environment, monitor changes over time, and make informed decisions to improve or protect the natural world. Environmental data can come from various sources, such as sensors, satellites, weather stations, and surveys. The whole process consists of data collection, cleaning and preprocessing, exploratory data analysis, statistical analysis, modeling and prediction, spatio-temporal analysis, data interpretation and reporting. The Evergreen project focuses on climate change monitoring, biodiversity, pollution control, and a sustainable environment to limit negative aspects of human activity. It aims to bring awareness of the importance of environmental data analytics and its practical reflection in daily activities. The used data set deals with aviation monitoring [7] [19].

The paper is organized as follows: Section 2 deals with the transaction log management to form a system covering historical data. Section 3 introduces a solution for optimizing log file management and access. Performance evaluation study is defined in section 4.

2. Transaction Logs Making Limited Temporal System

In database systems, UNDO and REDO are key components for managing transaction consistency and recoverability to ensure the database will always remain in a consistent state even in case of failure [2] [10]. UNDO tablespace is responsible for rolling back changes made during a transaction. In principle, it stores old versions of the object states in the form of a change vector. If there is an attempt to perform an Update or Delete operation, the original states before the modification are stored in the UNDO segment. If the transaction is ended successfully (by reaching COMMIT), UNDO data are no longer necessary, except for the attempt to get an isolated data image. Vice versa, if the transaction is rolled back (either due to the failure or manual user decision), the database system uses UNDO to revert all covered operations by the transaction. Thus, it is directly related to transaction support and management. UNDO data are stored in the database, in the physical UNDO tablespace.

REDO logs are used to store and save changes made to the database to ensure the durability and recoverability aspects of the transaction definition. REDO log is primarily stored in the memory for direct and fast access, however, before approving the transaction, particular transaction REDO log data must be moved to the physical repository – database – to ensure its availability after the system fails or collapses. Online REDO log files are formed into a set and are used in a sequence–cyclic manner.

As evident, both structures participate in data management during transactions. In case of changing data, a change vector is composed to store the original state, as well as the list of performed changes. UNDO is used for rolling back uncommitted changes, while REDO for recovering successfully ended transactions after the failure. From the data protection point of view, REDO is preferred to be stored for a long time, in contrast to UNDO, which is directly related to the existence of a particular transaction. One way or another, such structures are temporary and

overwritten over time, resulting in an inability to preserve historical data reliably during the whole time spectrum.

The concept of archiving partially solves that. If the *Archivelog* mode is enabled, filled REDO data are moved to the archive repository before rewriting. It enables wider recoverability, but also the ability to build historical data images. Fig. 2 shows the architecture of the archiving. Online logs are cyclically used for active transaction data. Before the file is overwritten, it is copied to the Archive repository by the Archiver (ARCn) background process.

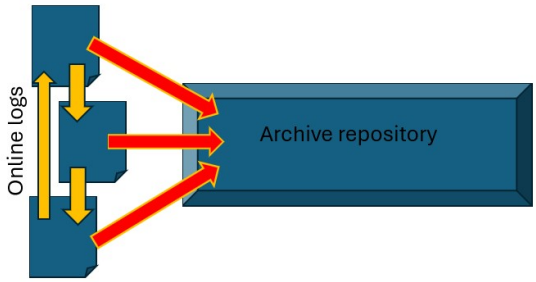

Figure 2. Archiving

2.1 Flashback

Flashback features were initially created as a transaction management extension in Oracle Databases. They provide historical states on various granularities – statements, tables or the whole database. Besides, it is possible to recover data based on the transaction logs - without the need to access traditional backup services. It uses online and archive logs to serve that. There are five available features related to Flashback technology [10] [11] [12]:

- Flashback query – gets the result set based on the data, as they existed at a specified timestamp or system change number (*SCN*) reference.
- Flashback table – reverts the whole table as it existed in the past. That is, the data themselves in the table are changed.
- Flashback database – reverts the whole database as it existed in the past.
- Flashback transaction – allows all the changes to be made by the particular transaction.
- Flashback table restore – recovers the already dropped table from the recycle bin.

There are three core limitations of the Flashback technology:

- There must be a sufficient amount of transaction logs to enable historical data. If even one element is unavailable, the entire operation will fail.
- The length of processing depends significantly on the amount of transaction data and the timestamp to which we want to restore the data. The deeper we go into history, the more complicated the whole process becomes, and it can take a very long time to get the result. At that time, the given structure is locked.
- All transaction logs must be sequentially scanned irrespective of the content. Even though there may be many transactions in the system and only one operates on the given data, the system cannot recognize it and, therefore, processes all data logs sequentially.

The above limitations significantly reduce the ability to process historical data in a conventional system via transaction management. Therefore, in the following subsection, we propose a new technique for transaction log indexing and referencing.

3. PROPOSED SOLUTION

Our proposed solution reflects a new architecture and approach to the historical data obtained by transaction log processing. It enables indexing and direct pointers to the relevant log block.

3.1 Indexing Transactional Data

Even though the transaction log can be divided into UNDO and REDO substructures, the transaction log typically contains many other structures and values. First, there is an identifier of the transaction, the operation performed, the SCN reference, and the change vector itself. Thus, a change vector is just part of that. SCN refers to the temporal position of the operation. It expresses a unique identifier, forming a sequential number for each change or transaction present in the database. It plays a crucial role in transaction management and consistency. Through that value, borders for the transaction operations can be easily identified. It is used for reverting operations, Flashback and recovery, etc. It is fundamental for consistent read operations. Although it does not precisely express time-based value, it can be associated with the transaction, which holds temporal time frame references. SCN can be obtained by invoking the following query from the V$DATABASE dynamic performance view:

```
SELECT CURRENT_SCN
  FROM V$DATABASE;
```

Query getting historical data of the data_tab table can be specified like the following:

```
SELECT *
  FROM data_tab AS OF SCN <specific_scn>;
```

Performed operation stored in the transaction log can be any *Data Manipulation Language* (*DML*) operation – Insert, Update, Delete and

Select. One might inquire how the Select statement can impact or modify the data stored in a database. The truth is that Select accesses data blocks and can physically release locks on data that have been used in already completed transactions.

Thus, it is evident that transaction log is not an ideal solution for retrieving past states. It contains many records that are not necessary for building historical data images, such as various housekeeping activities. Moreover, the initialization point is always the current state; it is impossible to start with a state valid in the past and find the state based on that starting point.

Our proposed solution deals with the transaction mapper, which flags the data with the transaction reference to be applied and held. The architecture of the solution is shown in Fig. 3. It consists of the core data flow by identifying three activities. Change operation is conventional. Data are primarily stored in the instance memory Buffer cache, from which the data are loaded into the database storage using the *Database Writer* (*DBWR*) process. All changes are simultaneously reflected in the online log repository using the *Log Writer* (*LGWR*) process. Similarly, conventional queries access memory Buffer cache, where the data reside or are loaded from the database in the form of the data blocks. A significant change is, however, made during Flashback queries. A new system table structure is identified, called Hist_data_ref_tab. It consists of four attributes – transaction identifier (*Tid*), referred (modified) object identifier (*Objid*), timestamp of occurrence (begin point of the transaction) delimited by the *Tdate* attribute and source of the ISA hierarchy (*RowLog*). *OnLog* value refers to the online log repository, while *ArchLog* points to the archive repository. Such architecture assumes, that an archive repository is enabled. The architecture of the solution is in Fig. 3.

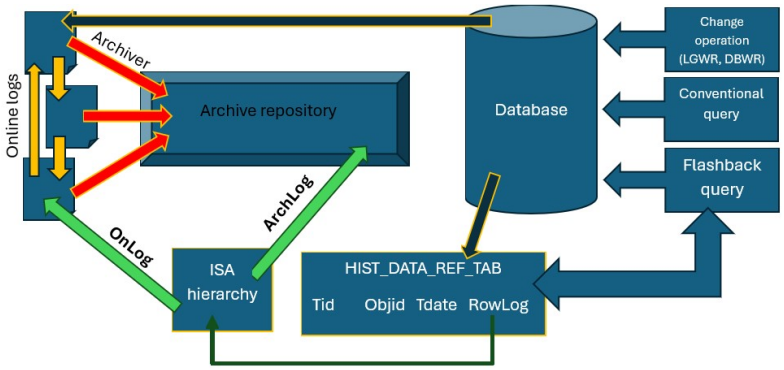

Figure 3. Flashback using ISA hierarchy + database operations

The whole system is depicted in Fig. 3. Red arrows express archiving, blue arrows refer to log management and log pointers are represented by green arrows.

Since there is an ISA hierarchy – *RowLog*, it is necessary to ensure data consistency associated with the archiving process. Namely, if the log data from the online repository are moved to the archive, a particular *OnLog* value is no longer valid and should be recalculated reflecting the new location. Therefore, an additional background process needs to be introduced. *LogMapper* background process is directly associated with the *Archiver* and is responsible for obtaining new *ArchLog* when the data are moved from online to the archive repository. In addition, this process is responsible for recording the change by performing an Update operation on the *Hist_data_ref_tab* table. Only when the entire process is complete, a corresponding online log can be released and overwritten. Fig. 4 focuses on the data and process flow of the transaction log management – Flashback query and internal log management and shift.

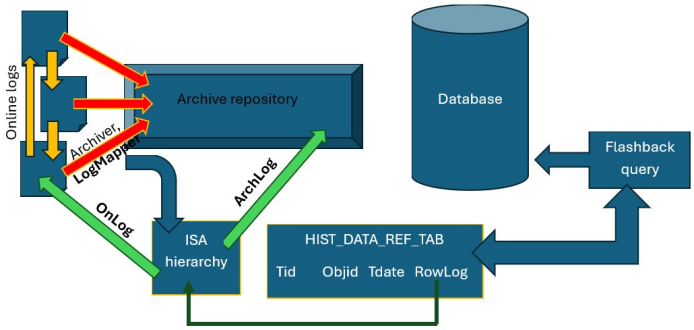

Figure 4. Flashback using ISA hierarchy + database operations

3.2 No ISA, Sequential Online Log Scanning

The above-proposed solution emphasizes ISA hierarchy as a root for the identifiers in online and archive repositories. It requires an additional *LogMapper* background process and object table holding ISA hierarchy references. Besides, it is necessary to synchronize the values to ensure that *RowLog* attribute values are unique. A simplified model is shown in Fig. 5.

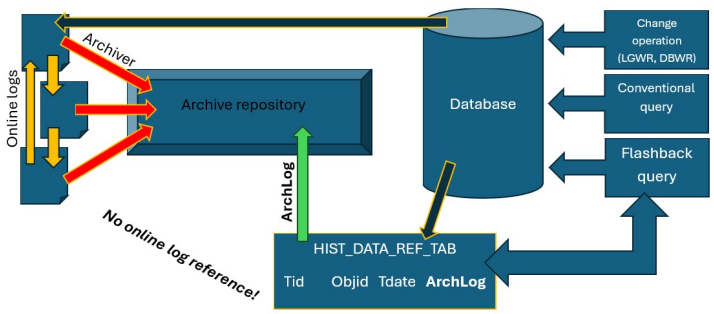

Figure 5. Updated architecture - ArchLog management – no ISA hierarchy

There is no ISA hierarchy, and no table structure holding references. *RowLog* values are reduced to point only to the archive repository. Thus, logical *RowLogs* are replaced by the *ArchLogs*. In *Hist_data_ref_tab*, the *ArchLog* attribute is placed, as well. Architecture differences are shown in bold (compare Fig. 3 and Fig. 5). Whereas there are no *OnLog* references available in the system, transaction online logs are scanned sequentially. It is based on the Oracle database configuration recommendation to set the online redo log size and global capacity appropriate for processed workload intensity [13] [14]. Furthermore, for security reasons, online logs are duplicated and stored in different repositories on different disks. Whereas transaction logging can be a bottleneck of the system, fast and reliable disks must be used [1] [2] [17] [18]. Based on these pre-conditions, it is assumed that the sequential scanning will be fast. Moreover, if there is a huge bunch of transactions, online logs are dynamically filled and moved to the archive repository. Note that this architecture follows the archive repository to be enabled using the following commands:

```
SQL> shutdown
SQL> startup mount
SQL> alter database archivelog;
SQL> alter database open;
```

3.3 Indexing Hist_data_ref_tab Table

From the performance perspective, the greatest pressure is on the temporal table (*Hist_data_ref_tab*) holding log references. To make access reliable and effective, it is necessary to index data based on the transactions, but mostly time reference.

Table indexing refers to an optional object associated with the table, which speeds up the data retrieval from the table. An index is a data structure that provides a fast way to look up values in a table. By default, the B+index tree is used but can be enhanced by various features and improvements [16], like single-column index, composite index, unique index type, function-based index or index based on virtual column calculated dynamically on-demand, reverse index, etc.

From the architectural point of view, there can be a clustered index, in which the rows are physically stored in the order of the index or a non-clustered index, which uses pointers (ROWID) to locate the rows [8] [16]. Typically, a table can have only one clustered index, which is not suitable for our temporal reference approach, because the data are operated and accessed either based on transaction reference or temporal perspective. Thus, two indexes will be developed to serve the workload requests.

There are 2 main disadvantages related to indexing:

- *Storage demands* – since the index is a physical object stored in the database; it requires additional disk space. It can be said that the index values are stored twice, in the database rows, but also in the index structure. If multiple indexes are developed, the value can be stored multiple times.
- *Slow change operations* – any write operation on the data, which manipulates with the index key values, must record change to the index, as well. Without that, a particular operation cannot be flagged as ended and the transaction must wait.
- *Maintenance* – indexes should be periodically optimized to limit migrated rows [8], in case of change operations. However, it is not a problem in our case. On the other hand, too-old data can be removed based on the archive repository *vacuum operation* [9]. In that case, the structure of the index can be changed rapidly, and the easiest way can be completely online rebuilt – the original index is dropped immediately after the new version is created. During the building operation, an original, not even quite efficient index is used.

Other used index types are bitmaps [12] [13], commonly used in analytical-oriented databases, in which data is static and does not change over time. Another index type is a hash index [12] [13], which divides the processed data sets into several buckets. The assignment of the row to the bucket depends on the result of the hash function. For the fast-evolving data, there can be a problem to find a function, which is robust and scalable [3] [4] [5] [6] [13]. Otherwise, the whole index can strongly degrade, resulting in the necessity to restructuralize and rebuild it, which can last too much time and would be required too often.

3.4 Technique of Post-Indexing

The concept of post-indexing [12] was introduced in 2022. It is based on extracting the process of data indexing to the separate transaction. It is intended for highly dynamic systems in which there is enormous pressure for change. Thus, it is perfectly suitable for the transaction log management in the temporal table (*hist_data_ref_tab* table).

The architecture of the post-indexing system is shown in Fig. 6. Any data changes (commonly Insert operations) are directly applied and stored in the *hist_data_ref_tab* table. However, changes are not directly indexed, because it could be too time and resource-demanding due to balancing, structure extension, allocating new blocks and extents, etc. [15] Moreover, each operation change requires such index operation, so the processing is often reduced to system maintenance work, not the changes themselves. Therefore, instead of direct indexing, a particular change is recorded to the flat memory structure. It serves as a temporary repository for the temporal data, which have not been indexed, yet. The introduced *Post-Indexer* background process is responsible for taking data from the flat structure and applying them to the indexes. It is done in a separate autonomous transaction, so the core transaction managing core data is not influenced. The data retrieval process is then extended by handling index data, but also flat data structure (which takes the same key attribute set as the associated index and ROWID locator) holding new object tuples, not covered by the index.

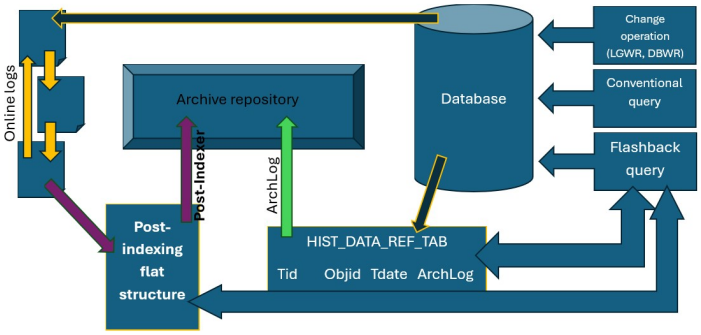

Figure 6. Post-indexing architecture

3.5 Disabled Archive Repository

The status of the database and used mode can be obtained using the following query.

```
SELECT LOG_MODE
   FROM V$DATABASE;
```

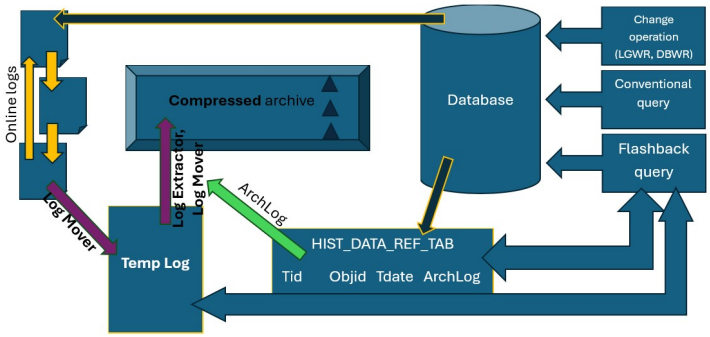

Figure 7. Own compressed archive in Noarchive log mode environment

The above query references the *v$database* dynamic performance view (stated in a From clause). The Select clause consists of the Log_mode attribute only. However, such a referenced table consists of dozens of attributes, which describe database structure and applied parameters [9]. The result of the query can be *NOARCHIVELOG* or *ARCHIVELOG,* expressing, whether the archive repository is enabled or not and background process *Archiver* is active or inactive. If the archiving is disabled, online transaction logs are not copied, instead, if they are marked as *inactive* (in case the particular transaction is ended), they are available for the consecutive rewritten. To serve the historical data management, own infrastructure for holding historical data in the form of reduced logs is created. Specifically, instead of a direct 1:1 copy of online transaction logs, they are temporarily moved to the specific repository. This space (*temp log*) is intended as a buffer for online logs so that transactions do not have to wait for the processing of expired logs to free up space. From that temp log repository, the newly introduced *LogExtractor* background process takes a stored log file and vacuums non-important transaction data, as well as references of the tuples, which do not need to be temporally monitored. Thanks to that, the size of the whole repository is less demanding. The process of copying extracted data from the temp log to the historical repository is operated and supervised by the *LogMover* process. Logical *RowLog* values and *hist_data_ref_tab* remain the same. The architecture of the whole solution is depicted in Fig. 7.

3.6 *Conventional Flashback Operation*

Flashback technology introduced in Oracle Database 10g [1] allows you to view historical data at a specific point in time, essentially "going

back in time" to query the state of your database at that moment. It is primarily used for recovery and auditing, but historical data can be obtained by the Flashback query, as well. It can be applied on *ARCHIVELOG* mode systems by accessing online and archive log repositories. Generally, whereas it primarily takes data from recent history, the archive repository can be omitted by enabling sufficient UNDO retention parameters [8] [16]. The following query expresses 1 hour (denoted by the seconds):

```
alter system set undo_retention = 3600;
```

Then, there are three values which can define the point, to which the database, table or query should be reverted:

• AS OF TIMESTAMP specifies temporal reference – date or timestamp based on the required granularity precision.

• TO_TIMESTAMP converts textual or date representation to the timestamp value expressing temporal reference.

• AS OF SCN refers to the specific SCN instead of a timestamp value.

The process flow is shown in Fig. 8. Since there is no direct reference to the objects handled, nor indexes, first, online logs are scanned, followed by the sequential scanning of the archive repository, up to the required timestamp or SCN respectively.

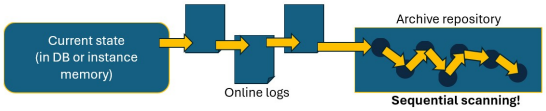

Figure 8. Conventional Flashback – sequential scanning

The query can look like the following:

```
SELECT *
  FROM DATA_TAB
    AS OF TIMESTAMP
       to_timestamp('2025-01-01 15:26:01',
                    'YYYY-MM-DD HH24:MI:SS');
```

A more complicated approach will be used if the temporal range is specified. In that case, the whole process takes all changes from the left border (*BD*) to the right border (*ED*). Interval representation (closed-closed or closed-open) must be also emphasized [11].

In that case, the query can then be specified like the following:

```
SELECT *
  FROM DATA_TAB
    VERSIONS BETWEEN TIMESTAMP
       to_timestamp('2025-01-01 15:26:01',
                    'YYYY-MM-DD HH24:MI:SS')
    AND to_timestamp('2025-01-03 11:03:09',
                    'YYYY-MM-DD HH24:MI:SS');
```

3.5 Exporting Transaction Logs

The proposed temporal solution dealing with the transaction logs must strongly emphasize physical infrastructure. Namely, *RowLog* values refer to the physical locations and positions on the disk storage. Similarly to pure indexing referencing the data using *ROWID* pointers, if the referenced solutions are changed, the whole index is marked as *UNUSABLE*. However, when dealing with the transaction logs, it would destroy the whole structure and all logs would have to be reprocessed by extracting relevant resources and data changes to index them in the second phase. It is important for data exports, as well as changing physical storage for logs, like moving repositories, changing disks, distributing logs across infrastructure or simply separating them to another repository. If any such activity is to be performed, it is necessary to hold references so that they are applicable even after a change of physical address. To serve that, *RowLog* values can be divided into two segments – the *base address* of the object storage and the *offset*. Of course, the block identification and the data position within it are also part of it. Thanks to that, only the base address is changed, but offset positioning remains the same. Fig. 9 shows the process flow of the export. Notice that any physical infrastructure can be logically performed as a conventional export.

Figure 9. Own compressed archive in Noarchive log mode environment

Fig. 10 refers to the import operation.

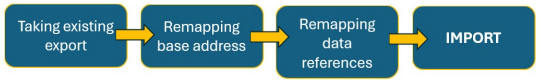

Figure 10. Own compressed archive in Noarchive log mode environment

4. PERFORMANCE EVALUATION STUDY

For the performance evaluation study, a real dataset of flight monitoring was used, which included spatiotemporal data tracking airplane locations and aviation assignments to the *European Flight Information Regions* (*FIR*). The dataset covered both planned and actual flight routes, monitored throughout the entire operation

— starting with the flight preparation, taxi, departure, and the flight itself, up to landing and parking. Temporal attributes related to the airplane's assignment to a particular FIR included entry and exit times. *The dataset contained 234,740 records from the European region, spanning the years 2017 to 2020.*

The environment for the evaluation study was set up on a server with the following specifications:

- Operating System: Windows 11 Pro, 22H2
- Processor: AMD Ryzen 5 PRO 5650U with Radeon Graphics, 2.30 GHz
- Memory: 2x 32 GB DDR-4, 3200 MHz, CL20
- Disk Storage: 2 TB, NVMe, read/write speed of 3500 MB/s
- Database: Oracle Database 23ai Free Release 23.0.0.0.0 – Production Version 23.4.0.24.05

Although various conventional systems can be used for the evaluation, performance was measured only on the Oracle database system. There are many reasons for that. Firstly, it is the most complex system, in which the Flashback technology operations were first presented, so the possibilities of its use and deployment are the widest. Secondly, significant changes and improvements can be identified through the last period related to the processing and performance in SQL [20], as well as procedural extension PL/SQL [20]. Thirdly, Oracle is a recognized partner of the EverGreen project [19]. However, although the performance is tested on only one system, proposed techniques and research results are generally applicable to any database system type, either in transactional or analytical format types.

For the performance evaluation study, the following models were evaluated. To make the study more representative and easier to handle, the following notations will be used – Tab. 1. The computational evaluation study handles 5 proposed solutions, compared to the conventional original Flashback (REF Model).

All proposed models were associated with three index sets, delimited by the following criteria:

- Transaction identifiers - *TransactionID*
- Temporal reference – *Tdate*
- Object reference – *objID*.

Three experiments were performed to highlight overall applicability, performance and limitations:

- Experiment 1 – conventional Select statement getting current valid data.
- Experiment 2 – change operation – Inserting new states to the temporal table.
- Experiment 3 – Flashback query.

Please note that the concept of temporality was reduced to handle historical data using transaction logging. Thus, even temporal database is stated, it is rather a logical approach, physically modelled using conventional tables and transaction logging.

Table 1. Evaluated model description

Designation	Referenced model type (architecture)
REF Model	Conventional Flashback system
Model **1**	Temporal transaction log model using *ISA hierarchy*, post-indexing **disabled**.
Model **2**	Temporal transaction log model, no ISA hierarchy, *online logs are sequentially scanned*, post-indexing **disabled**.
Model **3**	Temporal transaction log model, no ISA hierarchy, *online logs are sequentially scanned*, post-indexing **enabled**.
Model **4**	Temporal transaction log model, *disabled automated archive repository* (NOARCHIVELOG mode set), post-indexing **disabled**.
Model **5**	Temporal transaction log model, *disabled automated archive repository* (NOARCHIVELOG mode set), post-indexing **enabled**.

4.1 Results

The computational evaluation study was divided into three experiments. *Experiment 1* deals with the data retrieval process. To handle that, there were two tasks – to get the current position of the airplane *(task 1*) and to get all flights currently assigned to the particular FIR (*task 2*). Tab. 2 shows the results expressed by the processing time. Values are intentionally expressed in percentages to highlight the shift and improvements. Each task was performed 1,000 times, results express average values.

Table 2. Experiment 1 results – processing time

Designation	Processing time [%]
REF Model	100.00
Model **1**	101.65
Model **2**	100.74
Model **3**	100.62
Model **4**	100.55
Model **5**	100.49

The obtained results are depicted in Tab. 2. The differences are minimal at the level of statistical error and performance deviations. Thus, it can be said that they are (almost) the same, regardless of the type of log management. It can be stated that the slight differences may be associated with online logging because other transactions and new data are continuously produced, so the online logs need to be processed, freed, respectively moved to the archive repository. However, regardless of the used architecture, obtaining the current states of

the objects does not interfere with the logs and obtains the content only from the direct conventional table.

Experiment 2 deals with the Insert operation. In that case, conventional data are produced and saved in the main conventional table, Historical data could be obtained using transaction logs. Tab. 3 shows the results. Processing time is evaluated; however, such a metric does not need to be sufficient to compare individual solutions. Sure, from the user perspective, it gives a clear resolution, because it shows the demands. However, it is also necessary to point to the processing costs, which reflect not only processing time but also all system sources, mostly delimited by CPU consumption, memory demands, disk storage, loading, swapping, etc. Both metrics are evaluated in percentage. One million conventional rows were loaded, getting positions of the airplanes and FIR assignments. Provided values express average values.

Table 3. Experiment 2 results – processing time and costs

Designation	Processing time [%]	Costs [%]
REF Model	100.00	100.00
Model **1**	105.21	107.68
Model **2**	103.78	104.30
Model **3**	102.42	105.94
Model **4**	104.55	106.14
Model **5**	100.98	105.09

4.2 Processing Time

The worst results were produced by Model 1, which needs to handle ISA hierarchy and proper mapping between online and archive logs by requiring an additional 5.21%. By removing the ISA hierarchy, additional demands, compared to the REF model are 3.78% for Model 2 and 2.42% for Model 3. It is worth mentioning that these two models require sequential scanning of the online log to get historical data, however, such a requirement extension is not depicted by the data insert operations. By comparing the last two models (Model 4 and Model 5), a significant difference can be identified, which is caused by the post-indexing technique. This is because the archive mode is turned off and the system must extract relevant data from online logs and perform its archiving. It is not just a simple copy of the content, but rather a sophistical shift - the removal of descriptive transactional characteristics and content that we do not monitor at the time. The processing time difference is 3.57%.

The cost metric provides analogous results compared to the processing time, except for the last two models (Models 4 and 5). Namely, the ISA hierarchy is the most demanding. First, it requires additional database storage to store and map logical *RowLog* references, mapping between *OnLogs* and *ArchLogs*, as well as recalculation in case of data shift. Model 1 requires an additional 7.68%. Models 2 and 3 use sequential scanning of the online logs, so the whole architecture can be reduced to refer to *ArchLogs* only. Mapping is not required. When the data are to be archived, the pointer to the new location is obtained and stored as *ArchLog*. Indexing transaction logs requires an additional 4.30% of costs. Post-indexing takes an additional 1.64% of costs. Model 4 and Model 5 can be characterized by disabled archiving. Thus, the whole transaction log management and relevant data extraction are done by introduced background processes. Additional costs are 6.14% for Model 4 and 5.09% for Model 5. Post-indexing provides a better solution, an improvement of 1.05%. Although Model 5 requires additional space for objects to be indexed, core transactions can be ended sooner, so the resource demands are lowered. Furthermore, the waiting time for indexing is also reduced. So, even though storage and process costs have increased, overall costs have been reduced by 1.05%.

Fig. 11 shows the results graphically.

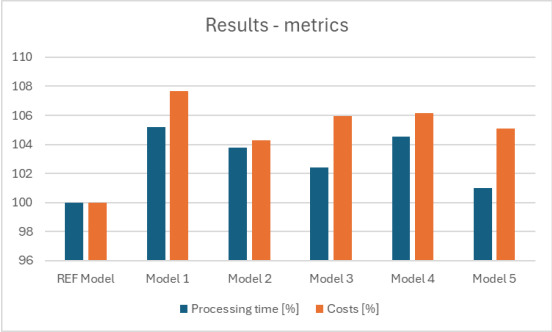

Figure 11. Experiment 2 - results

The last evaluation (Experiment 3) deals with data retrieval by applying transaction logs. In that case, the specification of the time reference was used. It was generated randomly for the whole range of the data, the same obtained value was then used for all models to refer to the same environment and conditions. Experiment 3 refers to a combination of two activities – aircraft monitoring, but also FIR assignments at a defined timestamp. Tab. 4 shows the results. The reference model (REF model) does not use transaction mapping using reference indexing, so all of them must be sequentially scanned. It means that even files, which do not operate particular data, need to be processed. Furthermore, since the FIRs are not the same in

terms of the space, or the number of assigned planes, searching for smaller FIRs is significantly more demanding. It is also important to mention that the FIR definition is not static, and its boundaries also change over time. The only applied rule for the REF model is the actual sorting of files in the archive in time. Thus, it is not necessary to go through all the files, but the processing is limited by the lower time limit. Fig. 12 shows the FIR definition for the European region.

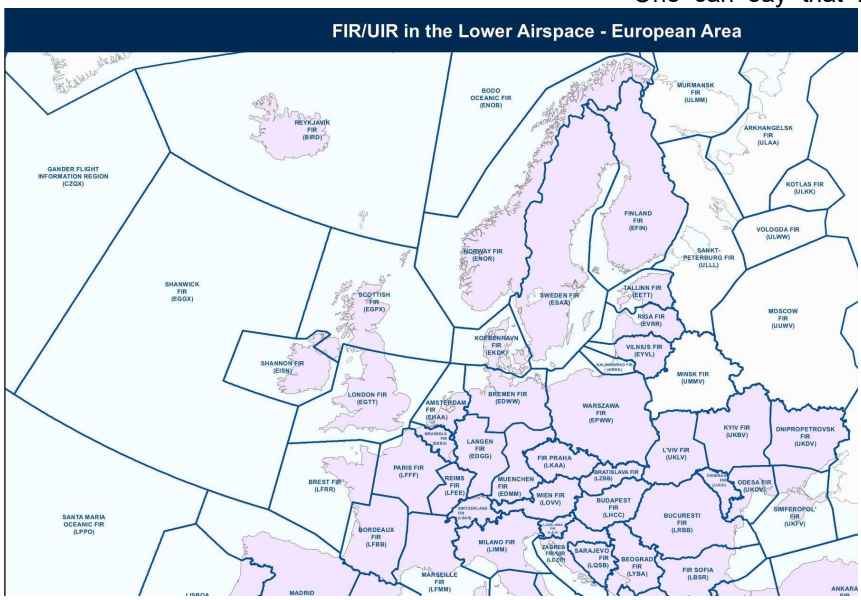

Figure 12. European FIR definition

Processing time: By using transaction log indexing, Flashback queries can be significantly accelerated. Primarily, only relevant data are accessed. It means, that if the log file does not contain changes on the required data, it is completely skipped. Change vectors are directly accessible, at the level of data blocks, respectively positions within them. The best performance reflected by the processing time demands was Model 4, which sequentially scans the online log, followed by searching in the proposed archive log, which is purified from transaction descriptive data. Since the logs are also block-oriented, the whole structure for the archiving is less demanding and requires less disk storage. The overall improvement of Model 4, compared to the REF model is 64.92%. The difference between Model 4 and Model 5 deals with the post-indexing, requiring sequential scanning of log data, which was archived, but has not been indexed, yet. Scanning reflects an additional 6.65%. Model 2 and Model 3 use an automated process of archiving, defined by the ARCHIVELOG mode database selection. Model 2 shows 56.08% improvement and Model 3 reflects 54.70% improvement. The cost of the post-indexing is 1.38%.

To conclude, own archiving reduces demands due to the size of the whole structure. This is a 12.41% improvement, expressed by the processing time reduction.

The cost of the processing is almost the same and expresses pronounced equality of dependence compared to the processing time. The only relevant difference can be seen in Models 4 and 5. The overall costs are higher. One can say that it is caused by the storage demand, which is not true because total size requirements are lowered. The increase in costs is due to the need to extract and reduce the transaction log data set. This requires obtaining relevant records and cleaning them from descriptive data. In this case, we assume that we monitor all data stored in the database over time. Precisely, there is no layer categorizing the data to be strictly conventional and history manageable.

Table 4. Experiment 3 results – processing time and costs

Designation	Processing time [%]	Costs [%]
REF Model	100.00	100.00
Model **1**	38.45	38.98
Model **2**	43.92	44.52
Model **3**	45.30	45.12
Model **4**	35.08	39.32
Model **5**	41.73	50.44

Fig. 13 shows the results of the processing time and costs graphically.

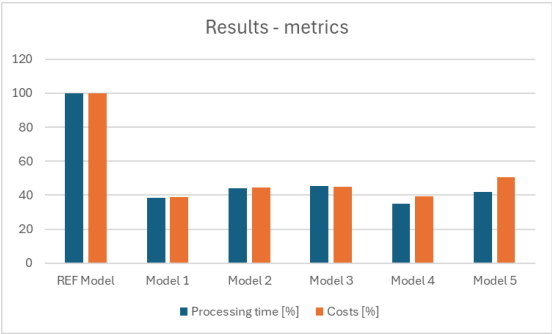

Figure 13. Experiment 3 - results

5. CONCLUSIONS

The conventional relational schema is characterized by storing current valid data. Since the relational paradigm is defined by the transaction support ensuring data consistency, atomicity, isolation and durability, change vectors are automatically stored in the online transaction logs. Even though they are intentionally used for the transactions themselves, they can be archived or relevant data can be extracted and stored in a specific repository. One way or another, making the transaction log data available brings the ability to collect historical data. Although it cannot benefit from the fully temporal system, because future valid data are not present in the system, by using existing data transaction resources, any data from the historical spectrum can be obtained.

This paper deals with the Flashback technology extension, enhanced by the indexes to navigate the relevant historical data. Online logs contain a large amount of descriptive data. Existing technology cannot assess which objects have been manipulated there. The proposed solutions use additional layers navigating the data by *RowLog* pointers. Various models were proposed and discussed to highlight their applicability and performance. In some cases, the mapping module was extended by the post-indexing, making the transaction confirmable sooner.

In the evaluation study, three experiments are discussed, dealing with the Insert statement, conventional query and optimized Flashback query. *Archivelog* mode of the system is primarily used to protect the database and make it recoverable after the failure. By using the proposed enhancements, the Flashback query can reduce processing time costs by up to 54.70%, if post-indexing is enabled. Vice versa, if the *Noarchivelog* mode is used, own extracted online logs are stored in a specific repository, to which the *ArchLogs* are pointed. In that case, processing time costs can be even more reduced, up to 58.27%. Thus, the performance benefits of the own archive solution, compared to conventional archiving is 3.57%. Among that, we evaluated costs, which are, however, closely related to the processing time.

During the future research, we would like to focus on the following topics:

- Extracting processed log data to attribute level (currently, we are dealing with object granularity for the logs).
- Temporal registration - not all data need to be historically evaluated. Therefore, we would like to propose commands, which will automatically set the archiving process.
- Strictly distinguish between individual query types and operations.

ACKNOWLEDGMENT

This paper was also supported by the VEGA 1/0192/24 project – Developing and applying advanced techniques for efficient processing of large-scale data in the intelligent transport systems environment.

REFERENCES

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

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

[3] T. Cunningham, “Sharing and Generating Privacy-Preserving Spatio-Temporal Data Using Real-World Knowledge,” In 23rd IEEE International Conference on Mobile Data Management, Cyprus, 2022.

[4] E. S. Chan, D. Gawlick, A. Ghoneimy and Z. H. Liu, "Situation aware computing for big data," 2014 IEEE International Conference on Big Data (Big Data), Washington, DC, USA, 2014, pp. 1-6, doi: 10.1109/BigData.2014.7004415.

[5] J. Heller, “Pro Oracle SQL Development: Best Practices for Writing Advanced Queries”, Apress, 2022

[6] S. Idreos, S. Manegold, and G. Graefe, “Adaptive indexing in a modern database,” In ACM International Conference Proceeding Series, 2012.

[7] J. Janáček and M. Kvet, “Shrinking fence search strategy for p-location problems,” In 2020 IEEE 20th International Symposium on Computational Intelligence and Informatics (CINTI), Hungary, 2020.

[8] D. Kuhn, S. Alapati, B. Padfield, “Expert Oracle Indexing and Access Paths: Maximum Performance for Your Database”, Apress, 2016

[9] D. Kuhn and T. Kyte, “Expert Oracle Database Architecture: Techniques and Solutions for High Performance and Productivity,” Apress, 2021.

[10] D. Kuhn and T. Kyte, “Oracle Database Transactions and Locking Revealed: Building High Performance Through Concurrency“, Apress, 2020

[11] M. Kvet, “Developing Robust Date and Time-Oriented Applications in Oracle Cloud: A comprehensive guide to efficient Date and time management in Oracle Cloud,” Packt Publishing, 2023, ISBN: 978-1804611869.

[12] M. Kvet and J. Papán, “The Complexity of the Data Retrieval Process Using the Proposed Index Extension,” IEEE Access, vol. 10, 2022.

[13] J. Lewis, “Cost-Based Oracle Fundamentals,” Apress, 2005.

[14] M. Malcher, D. Kuhn, “Pro Oracle Database 23c Administration: Manage and Safeguard Your Organization’s Data”, Apress, 2024

[15] B. Rosenzweig, E. Rakhimov, “Oracle PL/SQL by Example,” Oracle Press, 2023.

[16] S. Rosnan, N. A. Rahman, S. Mohamed Hatim and Z. H. Ghul, "Performance Evaluation of Inverted Files, B-Tree and B+ Tree Indexing Algorithm on Malay Text," 2019 4th International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE), Kedah, Malaysia, 2019, pp. 1-6, doi: 10.1109/ICRAIE47735.2019.9037757.

[17] W. Schreiner, W. Steingartner, V. Novitzká, “A Novel Categorical Approach to Semantics of Relational First-Order Logic,” In Symmetry-Basel, vol. 12, issue 10, MDPI, 2020.

[18] W. Steingartner, J. Eged, D. Radakovic, V. Novitzka, “Some innovations of teaching the course on Data structures and algorithms, ” In 15th International Scientific Conference on Informatics, 2019.

[19] Erasmus+ project EverGreen dealing with complex data analytics: https://evergreen.uniza.sk/

[20] Oracle 23ai: https://www.oracle.com/database/23ai/

Michal Kvet is an associate professor at the University of Žilina, Faculty of Management Science and Informatics, Univerzitná 8215/1, 01026 Žilina, Slovakia. In his research, the focus is on temporal databases, database performance, indexing and access optimization. Email: michal.kvet@uniza.sk, ORCID 0000-0003-3937-7473.