Concept of Select Unlocking Optimization

Michal Kvet
Department of Informatics, Faculty of Management Science and Informatics
University of Žilina
Žilina, Slovakia
Michal.Kvet@fri.uniza.sk

***Abstract*—Ensuring data integrity stored in a database requires change encapsulation in a transaction. Transaction maintains consistency by shifting the database from one consistent state to another, protected by atomicity, isolation, and durability. This paper focuses on the isolation and parallelism allowing distributed access to the data tuple across multiple transactions. Access to the data tuple is marked by the locks applied on the row level. Locks can be shared expressing data retrieval process or exclusive intended to change the values of the columns. The general locking rule takes the lock before the operation and releases it just after the operation or at the end of the transaction. Oracle database, however, does not release the lock physically, whereas the data block access would be necessary to be reloaded during the transaction approval. Therefore, this paper aims in optimizing the process of unlocking data before the data retrieval process using multiple strategies.**

Keywords—database transaction; unlocking; performance; data retrieval

I. INTRODUCTION

The main benefit of the relational paradigm and relational database processing is just the integrity, protected by transaction management. The transaction is an inseparable part of the data processing by shifting the database from one consistent state to another, which applies consistency, as well. It can consist of multiple operations inside, which are either completely accepted or the whole transaction is rejected [1] [2] [9].

Transactions and overall database processing are commonly placed in a complex environment, forcing the system to provide access to the data tuples in parallel, to optimize the performance and limit any waiting operations. On the other hand, whereas integrity should be ensured, no data can be lost, no change can be hidden, or no change can be refused after approval. To establish a parallelism option in databases, access to the data tuple must be marked and associated with the particular transaction, invoking the operation. For those purposes, transaction data locks are applied [9] [14]. If the data manipulation operation is to be executed (Insert, Update, Delete or Select), locks are applied on the row level granularity, while any structural change takes the whole object lock [5]. In principle, shared and exclusive locks can be present, depending on the operation to be applied on the row – whether the data are only obtained for the data retrieval process and evaluation or if there is an attempt to change to the content of the stored data. Individual locks influence the ability to access the tuple in parallel, as well as techniques for building a consistent state.

Among the transaction and individual operations, it is critical to set the proper rules for applying and releasing locks. Before any data touches, appropriate locks must be applied [3] [4] [6] [9]. However, what about releasing them? Either from a logical, but also a physical point of view? Well, releasing is not done later than at the end of the transaction. In principle, it must take all the blocks, which were operated in the transaction and wipe the used locks there. To do that physically, it would require taking the processed block set and proceed them sequentially. But there can be a significant performance issue [1] [8] [9] [10]. Namely, blocks do not need to be present in the memory due to the cache size demands and overcrowding [12] [20]. It, therefore, consequences in the necessity to load the block from the physical database storage once again for maintaining locks. It would naturally cause additional costs and I/O operations. In a reality, database systems use a different approach and postpone physical lock release to the next block access. It has, however, many negatives. Firstly, past information about the transaction's existence must be stored [11] [12]. Secondly, a transaction does not release a block at once, but it depends on the access [11] [12]. Thirdly, even after the instance restart, locks are still present in the database, consequencing their transfer across the exports, if they are done as a physical data file copy [15]. Fourthly, before attempting to obtain the lock on the data row, the system must evaluate, which locks should be applied as they reference active transactions and which ones are dummies referring to already-ended transactions [17]. It brings additional demands and costs. Moreover, in principle, irrelevant data are associated and stored inside the data blocks, aren´t they? It means that storage demands are not optimal. However, the main disadvantage of the entire lock management is based on transactions, which must be created dynamically as a result of lock processing and thus of data layer changes. Namely, when the block content is changed (lock is released), such a request must be recorded in the transaction logs. And thus trigger and invoke a new transaction, even if it is only about accessing data, without the need to change it. It requires proper categorization and normalization [18] [22] [23].

This paper aims to provide an additional layer treating the locks after the transaction end, without the necessity to reevaluate the locks associated with the row level granularity

stored in the block, as well as the necessity to reclaim new transactions.

This paper can be divided into three parts. The first part deals with the existing solutions, streams, and state-of-the-art. The second part provides the overview of the proposed solution, supervised by the additional extensions. The third part deals with the performance evaluation and optimization strategies limiting the data retrieval process in database transactions. Besides, the paper is structured into sections as follows: Section 2 deals with the transaction definition and ACID property aspect summary. Section 3 deals with the transaction log structure by emphasizing UNDO and REDO structures. They are critical in forming consistent data images during retrieval. Section 4 emphasizes the current approach of locking by pointing to the Shared and Exclusive locks, as well as limiting Shared locking in the Oracle database by spreading the wide ability of parallelism management [9] [21]. Finally, section 5 deals with the proposed solution and its enhancements, which are performance studies and evaluated in section 6.

II. TRANSACTION DEFINITION, ACID

A database transaction is formed by a set of statements encapsulated inside. It forms the core element of the database system processing by shifting the database to a new consistent state. The transaction is responsible for maintaining integrity by accepting all the constraints defined for the whole database, and table level, as well as applied for individual attributes or data groups. A transaction can be accepted and made durable only if all the constraints are passed, otherwise, the whole transaction is refused and changes rolled back.

The transaction is defined by these four aspects (ACID) [7] [8] [9] – atomicity, consistency, isolation, and durability. They are mainly protected by transaction log files. Atomicity is associated with the inability to process only part of the transaction. The whole transaction is either completely approved or rejected as a common unit. Consistency is related to applying integrity constraints and rules, not later than before transaction approval. If any rule violates, the whole transaction is refused, as stated in the atomicity option. Durability ensures the possibility of restoring the approved transaction after a crash.

Isolation is critical in terms of locking. Each performed operation in the database system must be time-referenced to the same moment. It is precise because of massive parallelism that data can change dynamically. In the analytical environment, the time reference is either the beginning of the operation itself or the entire transaction. Therefore, several situations can arise from the point of view of locking. Ideally, from the beginning of the operation, or the transaction, no changes on the relevant data blocks were performed, so the read operation provided as the data image is consistent and correct. Thus, it can be directly processed and included in the result set. In the second case, the lock is not applied over the record, but the data are not processed correctly in time and thus their status is more recent, compared to the required value. In that case, it is necessary to reconstruct the state as it existed at the defined time. However, such a state is not stored directly in the database, so it is necessary to access the change vectors of the transaction and use them to obtain the state in the past. For active transactions, this log definitely exists. If the transaction has already been terminated, the transaction log can be already overwritten and a Snapshot Too Old exception [9] [13] [16] occurs and the operation is canceled. In general, locks can be applied over the record. In that case, it is necessary to construct the historical state regardless of locks and active transactions on the object.

Thus, to construct any data image, relevant data blocks must be identified, followed by building a consistent image using transaction logs. Section 3 deals with the log structure, UNDO and REDO logs.

III. UNDO, REDO, AND FLASHBACK DATA ARCHIVE

The key to multi-versioning and building consistent data images is the existence of the UNDO. Each change operation creates a change vector, which specifies the original data row content (if exists) stored in the UNDO tablespace and content after the operation represented by the REDO structure. UNDO is stored in the database storage and is part of the read consistency. Vice versa, REDO is primarily stored in the memory for the active transactions. It aims at the durability of the approved transactions by copying online redo log content from the memory to the database storage not later that at the end of the transaction [9] [10] [11] [14].

UNDO information is applicable in two circumstances – multi-versioning support or transaction rollback. One way or another, it primarily focuses on reverse data operations. UNDO generation cannot be bypassed, as it protects the database, only direct-path data loading can navigate the processing for the physical storage by shifting the pointer to the last associated block - High Water Mark, if the loading is done successfully.

To protect the database and make the ability to get consistent data images for a longer period, the archiving process has been introduced. Transaction logs are primarily overwritten if the transaction becomes inactive. However, for dynamic systems with huge amounts of small transactions, it would be infeasible to reconstruct the historical image, as there is high pressure for the online logs and soon after the transaction approval, particular log would be overwritten. To prevent log loss for the UNDO reconstruction opportunity, it is advisable to copy the content of the log into another – archive repository. The architecture of the archiving is shown in fig. 1.

The algorithm for the tuple state reconstruction is summarized in the following steps:

1. Getting current data images obtained from the data block (located in the database or memory).
    - If the block is not currently in the instance memory, a free memory block must be identified and the relevant block loaded into the memory from the database storage.

2. Applying online UNDO data to construct historical data images using AS-OF version query definition.
3. Applying archive UNDO data if the previous step does not meet the time requirements referring to the start point of the operation or the whole transaction, based on the settings.

If any of the above steps failed, the ORA-01555 Snapshot Too Old exception is raised.

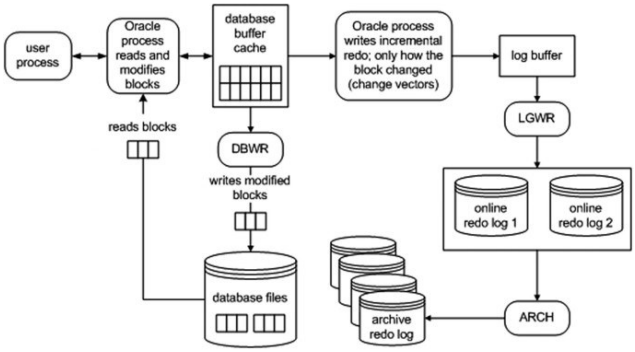

Figure 1. Log management [9]

Thus, to construct a consistent data image, all the required transaction log pieces must be present.

In the environment of consistent data image construction and change management, two locks can be identified, differing in the access method and assumed data manipulation operation. Section 4 deals with transaction locking.

IV. LOCKING

Locks are mechanisms limiting and regulating access to shared resources. Database systems can use various precisions for the locking granularity, like SQL Server taking page-level locking, resulting in the inability to perform concurrent Insert statements, even if they are not blocking each other based on the unique constraints (primarily expressed by the primary key). Informix used a predefined number of locks, which could not be exceeded, later. The value was configured before starting the database as a static parameter. In the past, applied table-level locking was also intended, however, it would be impossible to access and table data in parallel. Oracle database uses row-level locking. The finest granularity is the column itself, however, this option would require too many locks. For example, if the table consists of 10 attributes, by invoking the Insert statement, at least 10 locks would be necessary to be placed, as well as stored physically. As a consequence, the data block would degrade to hugely manipulate the locks.

Besides, various lock strategies are discussed, pointing to the lost updates [9], pessimistic [9] [15] and optimistic [9] [15] locking using version column [12] or checksum [16] [19].

Based on the SQL norm, two lock types exist. A shared lock protects the tuple from executing change on that row. It is associated with data retrieval (Select statement). Several Shared locks can be applied for the same row allowing parallelism in the data access and querying. An exclusive lock is associated with the change operations limiting the parallel access to the data tuple. By using an Exclusive lock, only one transaction can operate the data in any manner (change and data retrieval). Fig. 2 shows the lock accessibility matrix.

	Shared (S)	Exclusive (X)
Shared (S)	Y	N
Exclusive (X)	N	N

Figure 2. Locking type matrix

In case of changing the structure of the table, an Exclusive table lock must be applied meaning, that other transactions on that table are prohibited.

Oracle database does not use Shared lock, at all. The data image is protected by the UNDO transaction data by building a consistent image [9]. Thus, the version of the data tuple is checked during the retrieval and if necessary, UNDO change vectors are applied to get the state as existed at the beginning of the operation or the whole transaction. Thanks to that, the lock management is simplified and the parallelism options can be wider spread.

There is, however, a huge performance issue related to the lock release.

The performance issue is related to the lock release. After the processing, a particular lock can be released anytime, based on the applicable rules or business logic, but it cannot be released later than at the end of the transaction. Data lock is associated with the type (Exclusive (X) / Shared (S)) and transaction reference. Thus, from the logical point of view, releasing an object is reflected by making it available for consecutive processing and changes. There can be a list of transactions waiting for the object using the FIFO approach, thus the next transaction can operate the data, generally.

The physical points and principles are, however, more complicated to ensure the consistency and the performance of the lock release management. Namely, to release the tuple, it is necessary to touch the relevant database block and vacuum the transaction lock reference. However, where is that block located? Ideally, it is placed in the memory Buffer cache, so the access is straightforward by identifying the block and changing its content (lock release). It, naturally, requires storing additional information in the transaction log, whereas the block content is changed. The more complicated situation, however, arises, if the particular block is not currently in the memory. In that case, it would be necessary to obtain the address of the row (ROWID), load the block into the memory, release the lock physically, record the change in the transaction log and finally, save the block physically into the database. As evident, such physical lock release could be too demanding requiring additional I/O operations and limiting the performance of the system – whereas the release is protected by the transaction UNDO and REDO structures, transaction approval can be done only if all the data are logged, as well as physical operations are successfully done. Therefore, database systems postpone the physical lock release to the next usage of the block meaning, that improper locks are still present in the database blocks. Moreover, when attempting to access the data tuple, it is necessary to apply the lock. Before that, however, expired row locks must be removed from the consideration

bringing additional demands. The data flow diagram is shown in fig. 3.

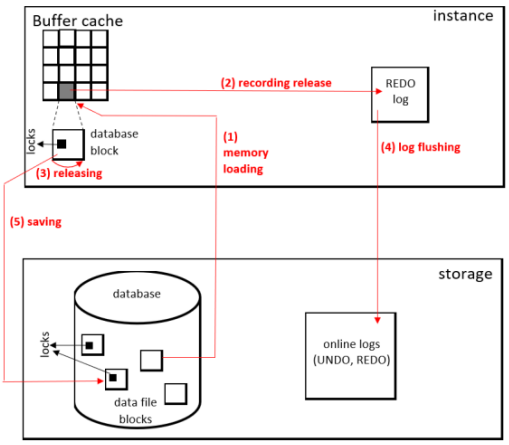

Figure 3. Data flow diagram

Section 5 provides a new approach to managing the data locks.

V. Proposed Solution

The proposed solutions discussed in this paper aim to identify locks applied for the tuple or generally applied for any object inside the data block. Namely, we use the block grouping approach in this paper. Thus, if the tuple is to be evaluated by identifying and limiting expired locks, the whole block is reconsidered, whereas the finest processed granularity of the memory Buffer cache is the block itself, thus the whole block must always be fully loaded for the consideration and evaluation.

To ensure the performance, the new structure needs to be introduced, dealing with the transaction reference list, which can be associated with the row tuple itself, or the block references can be used. It is named the Lock Tracker module and it consists of the object tuple identifier, transaction, and applied locks. Its internal data structure can vary, depending on the linear linked list or depending on the applied priority rules forming B+trees. Generally, random distribution in the dynamic array can be used, as well, however, if a massive number of transactions is present, identifying the ended transactions would require scanning the whole structure.

The first solution (**SOL_B+pure_tuple**) is associated with the primary key or unique constraint, which identifies any data row, any tuple stored in the table. It uses the rule of the database, that each table of the relational database uses a primary key for object identification. If this were not the case, an artificial identifier would be introduced in the form of a sequence of records or physical addresses. One way or another, the B+tree index would be present, consisting of the key for traversing through it. The leaf layer stores the key value and 10-byte value addressing the position of the row in the physical database storage layer – data file, data block, and position of the row inside the block. Besides, in our proposed solution, we have added there a list of locks applied for the tuple. It takes the transaction reference list, pointing to the transaction log definition and applied lock type. Expired locks are marked by the boolean flag. The architecture of the solution is shown in fig. 4. Lock Tracker stores the tuple reference (ROWID) and transaction reference list. In this approach, it is shaped by the linear linked list, where each element stores the transaction identifier, lock type, expiration flag, and timestamp, when the lock was applied. Transaction start and approval timestamps are not present, whereas they can be obtained in the online transaction log segment.

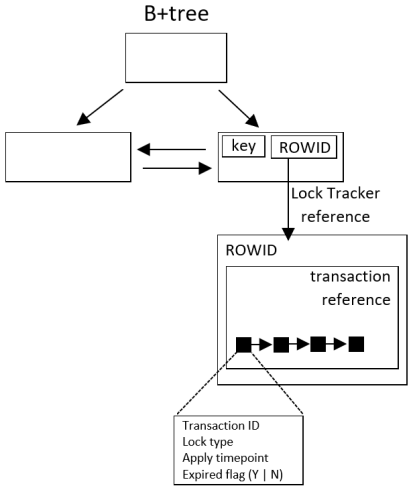

Figure 4. Solution architecture

The enhancement of the solution (**SOL_B+priority_tuple**) introduces the priority in the transaction reference list. Namely, expired transactions are put in the first part of the transaction reference list. Thanks to that, the search starts with the first element and can be stopped by reaching the first active transaction in the list. A proprietary, transaction reference list is divided into two parts – expired and active transactions, however, inside the structures, data are randomly distributed.

Among the solution specified above, two other enhancements have been evaluated and considered. They aim on using priority inside the transaction reference list to optimize and make easier transaction identification. For this perspective, two solutions were identified – the transaction reference list was B+tree oriented based on the transaction start timestamp (**SOL_B+time_tuple**), or the priority of the transaction was set based on the total number of applied locks (**SOL_B+locknum_tuple**). Namely, it is assumed, that expired transactions with a higher number of applied locks should be prioritized.

Another development and research stream was associated with block management instead of the tuple. Whereas the finest processed granularity to be transferred between the database and memory is the block itself, it is useful to process the whole block instead of tuple management. Thus, by accessing the block, all expired transaction locks must be identified and removed. To do that, the Block extractor data structure has been introduced as an extension of the Lock Tracker reference. Three physical data structures and models

were applied for the Block extractor module. **SOL_B+LL_block** introduces a linear linked list for the tuple mapping to the data blocks. **SOL_B+bitmap_block** uses a bitmap index structure on the block level characterizing the x-axis. The object identifier (primary key) is defined by the y-axis. The content is a boolean value. Note, that there is only one block reference for each bitmap column. The last solution (**SOL_B+stitching_block**) does not introduce a separate storage structure for the data block and tuple association, instead stitching in the B+tree layer is used. Thus, instead of sorting the data based on the index key on the leaf layer, linking is done based on the block references.

A. Lock vacuuming

There are many circumstances when expired locks can be removed. The whole process aims to optimize I/O operations and minimize costs during the processing. Therefore, we have identified the following operations, which can be preceded by the lock reevaluation:

- Block is to be saved by moving from the memory Buffer cache to the database using the Database Writer background process.
- CheckPoint operation, which completely frees memory Buffer cache by saving dirty (changed) data blocks to the database.
- Reloading from the database before evaluating in Buffer cache.
- Existing data blocks are present in the memory, identified by the direct pointer from the Lock tracker.
- Dynamic remapping – loading a small number of blocks into the memory on demand and vacuuming the locks (mostly operated during weak database workload).

VI. PERFORMANCE AND DISCUSSION

The environment used for the performance evaluation was characterized by the multi-tenant Real Application Cluster container database. The used database system was *Oracle Database 21c Enterprise Edition (Release 21.0.0.0.0 – Production.* The server parameters were delimited by the following values:

- Processor: AMD Ryzen 5 Pro 5650 2.3GHz
- Operating memory: 64 GB, DDR4 3200MHz,
- Disc storage: 2TB PCIe Gen3 x4 NVMe v1.4, reading 3500 MB/s, writing 3300 MB/s.

The air transport monitoring data set was used for the analysis, consisting of the planned and real routes of the airplanes, enhanced by the various parameters characterizing the flight conditions, like speed, height, or positional data. Besides, each airplane was monitored by assigning its position to a particular region – FIR (Flight Information Region). A temporal data model using synchronization groups [13] was used. Validity was associated with the FIR entry and exit time with the second granularity precision, defined by the Oracle Data data type. The data snippet of the FIR assignment is shown in fig. 5.

```
"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"
```

Figure 5. Evaluated data structure – Flight information region (FIR) temporal assignment

European regions for the FIR definition are shown in fig. 6.

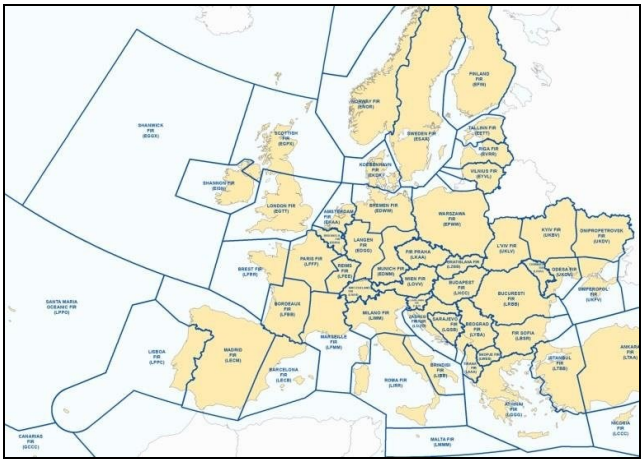

Figure 6. Flight information region (FIR) map in Europe [24]

The evaluation study was focused on the data loading operations – inserting new data object states to the database – data obtained from the airplanes separately or bulk collected within the data catcher and group module. Then, the Update operation was evaluated, pointing to the difference between the planned expected route and the real route. Additionally, an Update statement was made by using data corrections if the communication fails or the data transfer was delayed.

The second evaluation stream was characterized by data retrieval, by which the processing was easier, whereas expired locks were not present, and it was no longer necessary to check the locks and identify active transactions during the data retrieval process. It must be also noted, that removing the lock requires physical data block change, which is always protected by raising new transaction by storing change vectors in the log.

To reach the complexity of the processing, data maintenance operations are also evaluated, which are typically launched during a weak workload. The activities there are mostly related to the CheckPoint and Remapping.

The results evaluated the processing time and costs of the processing. For declarative purposes, reached values are in percentage. Each evaluation study was performed 10 times and the results express the average values.

Fig. 6 shows the results for the costs, which consider the server resources, processing time, data storage, CPU, I/O operations, and overall used sources. Costs are the internal indicator of server resource utilization. The reference for the evaluation is the original solution with no explicit lock release management and cleaning operation. For the Insert statement,

additional demands can be identified, ranging from 2% to 11% depending on the structure and priority management. Namely, recording and prioritizing transactions based on its start point requires an additional 7%. This indicator, however, does not provide sufficient power, whereas long-run transactions with low lock number can be present. Therefore, the more relevant solution is associated with recording the lock number applied for the data tuples by making that value a priority for the B+tree definition. Thus, the lock number is treated as a key of the B+tree and locator through the balancing. For the lock number recording, 11% can be identified for the direct separate management and 12% for the bulk management, which aggregates the locks and applies the change just after the bulk operation itself. By comparing the direct Insert statements and bulk operations, it can be concluded, that there is no significant difference on the performance. When dealing with the blocks, an additional 3 up to 5% can be identified, just expressing the management inside the block extraction module and grouping ROWIDs to the same block, reflecting the data migrations, as well, if the Update operation extends the storage demands and the original block can no longer serve the state due to its capacity. As evident from the results, for the Insert operation, an additional 19-20% of costs are present for the bitmap management of the block definition. By analyzing the costs and overall structures, it can be concluded, that it is caused by the bitmap, which must be reconstructed by adding a new extent (a set of blocks are newly associated with the table), requiring extending and recalculating the bitmap matrix content. The rest operations – Update (comparing expected and real route and management of data corrections) lower the demands up to 15% for the block management using stitching. Namely, Update statements benefit from the lock expiration management by lowering the demands for active transaction identification. Only pure B+tree brings slightly higher demands, whereas the transactions are randomly distributed across the list with no priority management and there is no specific differentiation between active and ended transactions, thus the list must be always fully scanned.

As the title of the paper denotes, it primarily aims on optimizing the performance of the data retrieval process, represented by the Select statement. Looking at the reached results, by applying operations and collecting expired transactions by removing outdated locks, significant benefit can be identified, lowering the demands up to 45% for block management. Pure tuple management lowers the demands up to 50%. By introducing the priority, demands can be generally lowered up to 59%, and the start point of the transaction takes 61%. The best solution was provided by the lock number management as a priority lowering the demands up to 61%. It can be concluded that the number of applied locks is the best indicator for marking the transaction. The more locks used in a transaction, the more prioritized it should be in the event of its termination.

Finally, the data block maintenance has been also evaluated, and managed by the CheckPoint and Remapping operations. Although additional demands can be identified, ranging from 5% through 8% up to 12%, the operations are done during the low workload of the system, therefore, it does not make much sense to optimize and limit the additional costs, mainly due to the fact, that in the case of the Select statement operations, the performance has increased radically.

Fig. 7 shows the costs additional costs in percentage by taking original lock release management as a reference.

COSTS		Original lock release management	SOL_B+pure_tuple	SOL_B+priority_tuple	SOL_B+time_tuple	SOL_B+locknum_tuple	SOL_B+LL_block	SOL_B+bitmap_block	SOL_B+stitching_block
			Tuple management using ROWID				Block management using BLOCKID extracted from ROWID values		
Insert	Direct separate	100	102	104	107	111	105	120	108
	Bulk	100	102	103	107	112	104	119	108
Update	Comparing expected and real route	100	101	97	95	93	92	87	87
	Data corrections	100	105	101	99	98	88	85	85
Data retrieval		100	50	41	39	36	55	54	55
Data block maintenance		0	12	8	8	8	5	5	5

Figure 7. Results - costs

The second evaluation stream was based on processing time. Fig. 8 shows the results, expressed in percentage, as well. These characteristics do not reflect the I/O operations, storage and resource demands, instead, the demands are just reflected by the total time required to complete the task.

For the Update statements, the system benefits from extracting the block, instead of using data row granularity. The reason is based on the data migrations, where the original row is moved to another block. However, by using the index, both blocks are memory loaded, thus the expired locks are automatically identified and removed. The processing time demands are even more decreased, compared to the total costs. Namely, for the data retrieval process, the best solution was obtained using lock number management, lowering the demands using 65%. The start timepoint of the transaction does not need to express the right priority, mostly if there is complex analytics on the data, so the data source is not huge, the complexity is reached by the calculations and data processing.

Inside the maintenance process, only 6% of demands are present for the block granularity processing, whereas there is direct access to the expired transactions list. Compared to tuple management, block references must be extracted, bringing an additional 1 to 2%.

The significant performance degradation of the processing time can be, however, identified for the solution using bitmaps. Namely, the additional demands have risen to 29% for direct Insert statements and 34% for bulk loading. Well, the bitmap cannot be easily extended and requires structure reconstruction and rebuilding. Although the bitmap storage

demands are low, as shown in fig. 7, processing time requires significantly higher values (fig. 8).

PROCESSING TIME		Original lock release management	SOL_B+pure_tuple	SOL_B+priority_tuple	SOL_B+time_tuple	SOL_B+locknum_tuple	SOL_B+LL_block	SOL_B+bitmap_block	SOL_B+stitching_block
			Tuple management using ROWID				Block management using BLOCKID extracted from ROWID values		
Insert	Direct separate	100	103	105	108	110	103	129	106
	Bulk	100	102	104	108	113	104	134	107
Update	Comparing expected and real route	100	102	98	94	92	94	95	89
	Data corrections	100	106	102	99	98	89	93	83
Data retrieval		100	48	39	38	35	48	71	50
Data block maintenance		0	10	7	8	7	6	6	6

Figure 8. Results – processing time

Processing time results in form of chart are present in fig. 9.

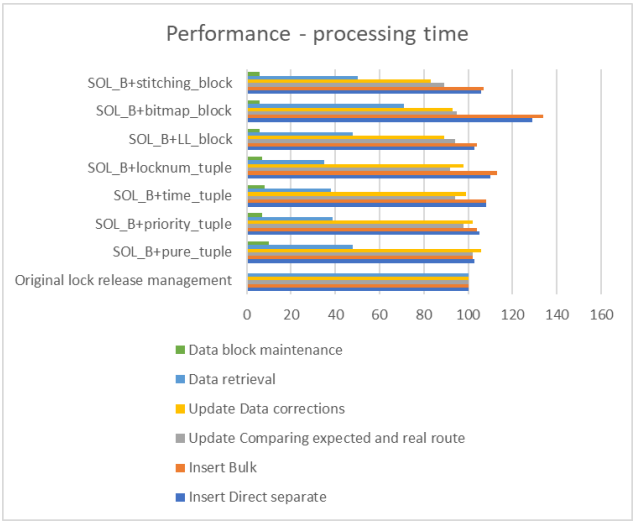

Figure 9. Results – processing time – graphical representation

VII. Conclusions

Efficiency in data processing is critical for the whole data management and is directly reflected by the performance of the whole information system. The massive parallelism for the data access is currently required, supervised by getting consistent data images based on the transaction UNDO logs. To ensure consistency and no data loss, access to the data must be properly controlled by the locks. In relational databases, based on the SQL norm concerning transactions, Shared and Exclusive locks are present for manipulating the data and updating them. Data locks are applied for the data tuple, generally to bring a suitable environment for parallel access and processing. These locks are stored within the data in the database blocks. Thus, before attempting to access the data, the particular block is memory loaded and particular locks are applied, if possible. The release of the locks is then applied based on the business rules, but not later than at the end of the transaction. The state-of-the-art solutions, however, apply for the lock release only logically, whereas it would require additional data block access and I/O operations. As a consequence, the loaded data block holds expired locks, which must be sequentially reevaluated to vacuum expired ones. The principle is based on touching the data, preceded by the lock consolidation. As stated, it brings additional demands, whereas the lock management cannot be removed or postponed without the integrity risks.

The proposed solution brings an additional layer of managing and securing data locks. It aims in vacuuming expired locks for already-ended transactions in a separate process. Thus, before writing the block from the instance memory to the database, transaction locks applied for the block are reevaluated. Moreover, the specific module for the transaction expiration recording is introduced, which uses the number of accessed and changed blocks as a priority level. Thanks to that, the number of expired locks is significantly lowered. It brings two main advantages. Firstly, disc storage demands are lowered by allowing a higher fill ratio of the block. Secondly, removing the expired lock operation before attempting to process the data row is shortened and optimized by the introduced transaction reference module. In this paper, pure tuple and block granularities have been analyzed, followed by the transaction list reference in B+tree using transaction identifiers, priorities defined by the start timepoint of the transaction, or the number of applied locks inside. Another solution was based on the bitmap structure, which, however, does not allow dynamic rebuilding by adding new data blocks.

The aim of future research will focus on transaction distribution and locking across multiple instances operating the pluggable database. Environment-applying table partitions can be characterized by various local indexes for each data fragment. It is assumed, that the proposed solution can reach even more significant performance benefits. On the other hand, it would require an additional synchronization layer recording expired transactions across the distributed ecosystem.

Acknowledgment

It was supported by the Erasmus+ project: Project number: 022-1-SK01-KA220-HED-000089149, Project title: Including EVERyone in GREEN Data Analysis (EVERGREEN).

Co-funded by the Erasmus+ Programme of the European Union

EverGreen DATA Analytics

REFERENCES

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

[2] Al-Sanhani, A.H., Hamdan, A., Al-Thaher, A.B., Al-Dahoud, A.: A comparative analysis of data fragmentation in distributed database. ICIT 2017 - 8th International Conference on Information Technology, Proceedings. 724–729 (2017). https://doi.org/10.1109/ICITECH.2017.8079934

[3] Cornejo, R.: Dynamic Oracle Performance Analytics. Dynamic Oracle Performance Analytics. (2018). https://doi.org/10.1007/978-1-4842-4137-0

[4] Dudáš, A., Škrinárová, J, Vesel, E.: Optimization design for parallel coloring of a set of graphs in the high-performance computing. In: Proceedings of 2019 IEEE 15th International Scientific Conference on Informatics, pp 93–99. ISBN 978–1–7281–3178–8

[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] Graefe, G., Guy, W., Sauer, C.: Instant Recovery with Write-Ahead Logging: Page Repair, System Restart, Media Restore, and System Failover, Second Edition. Synthesis Lectures on Data Management. 8, 1–113 (2016).

[7] He, Q., Zhang, F., Bian, G., Zhang, W., Duan, D., Li, Z., Chen, C.: Research on Data Routing Strategy of Deduplication in Cloud Environment. IEEE Access. (2021).

[8] Jin, D., Chen, G., Hao, W., Bin, L.: Whole Database Retrieval Method of General Relational Database Based on Lucene. Proceedings of 2020 IEEE International Conference on Artificial Intelligence and Computer Applications, ICAICA 2020. 1277–1279 (2020). https://doi.org/10.1109/ICAICA50127.2020.9182496

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

[10] Lew, M.S., Huijsmans, D.P., Denteneer, D.: Optimal keys for image database indexing. In: Del Bimbo, A. (ed.) Image Analysis and Processing. Lecture Notes in Computer Science, vol. 1311, pp. 148–155. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63508-4_117

[11] 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

[12] 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

[13] Kuhn, D., Alapati, S.R., Padfield, B.: Expert Oracle Indexing and Access Paths. Expert Oracle Indexing and Access Paths. (2016). https://doi.org/10.1007/978-1-4842-1984-3

[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] 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

[16] Kvet, M.: Database Block Management using Master Index, FRUCT 32 conference, 9-11 November 2022, Finland

[17] Kvet, M.: Covering Undefined and Untrusted Values by the Database Index, Information Systems and Technologies, Lecture Notes in Networks and Systems, 2022, ISBN: 978-3-031-04828-9

[18] Kvet, M. and Papán, J.: The complexity of the data retrieval process using the proposed index extension, IEEE Access, 2022.

[19] Kvet, M.: Relation between the Temporal Database Environment and Disc Block Size

[20] Qian, Z., Wei, J., Xiang, Y., Xiao, C.: A Performance Evaluation of DRAM Access for In-Memory Databases. IEEE Access. 9, 146454–146470 (2021). https://doi.org/10.1109/ACCESS.2021.3123379

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

[22] Rolik, O., Ulianytska, K., Khmeliuk, M., Khmeliuk, V., Kolomiiets, U.: Increase Efficiency of Relational Databases Using Instruments of Second Normal Form. 221–225 (2022). https://doi.org/10.1109/ATIT54053.2021.9678605

[23] Schreiner, W., Steingartner, W., Novitzká, V.: A novel categorical approach to semantics of relational first-order logic. Symmetry 12(1584), 2020 (2020)

[24] https://www.researchgate.net/figure/Flight-Information-Regions-in-Europe_fig3_328927722