Enhanced Data Locking to Serve ACID Transaction Properties in the Oracle Database

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

***Abstract*—Relational databases are characterized by precise data models, formed by the entities and relationships. Integrity and consistency are maintained by the transactions, which ensure transfer from one consistent state to another. Consistency is ensured at the latest at the end of the transaction itself. SQL norm specifies two lock types applied on the data row level to serve the consistency in the parallel data access. However, Oracle Database uses a different approach relying on the transaction logs to provide historical consistent data images. In this paper, Reservable attributes are discussed, introduced in Oracle 23c version by focusing on the performance, limitations and general issues.**

I. INTRODUCTION

Relational databases were introduced in 60ties of the 20th century. Immediately after their introduction, several variants and efforts for certification and codification were present. A relational database is formed by the set of entities and relationships between them, by focusing on the data value precision and accuracy. This is done by the transactions, which ensure the shift from one consistent state to another, by checking the integrity rules. [11]

Even after decades, the relational paradigm is still widespread, primarily operated by the SQL language, which is a non-procedural type, so the user only specifies, what data, shapes, and formats should be provided, but not, how to reach them. Thus, the access to the data and result set composition is left to the database systems and internal techniques, mostly delimited by the optimization, indexing, and physical repository. The relational paradigm is mostly related to storing current valid states by replacing original states during the change (Update operation), called the conventional system [11]. Currently, the focus is done on the temporal databases, which can hold not only current valid states, instead all the states associated with the object are stored. Thus, there are historical states valid in the past, but also states, that will become valid later as future plans. The temporal sphere, mostly defined by the validity, can be defined on various precisions and granularities, either for the object, attribute, or synchronization group level [6] [8].

Temporal relational databases form the core part of the information technology used for intelligent information systems, analytics [3], teaching [14], machine learning techniques or decision-making, and critical systems [10] [11]. The reliability of the stored data is protected either by the data model itself, but mostly by the integrity rules and transactions. Data transparency is checked continuously during the transaction run, but before reaching the transaction approval point, all the integrity rules must be passed. Otherwise, the whole transaction is refused and the original data remain.

The data access and change operations are protected by the data locks applied for the table tuple. General SQL norm defined *Shared* lock for the data retrieval and *Exclusive* lock for the change operations. If the row is to be accessed, *Shared* lock must be applied to ensure nobody can change the content of the row during the Select operation. Vice versa, *Exclusive* lock is used to ensure nobody changes the data in parallel, nor the tuple is accessed and retrieved, generally [4], [5], [9].

For this paper, the Oracle Database environment will be used, delimited by the release bundle Oracle 23c Free, Developer Release Version 23.2.0.0.0, introduced in April 2023. There are several reasons to select Oracle Database and not to focus on other database system variants, generally. Firstly, it is most powerful solution in relational databases. Besides, it is the most robust and complex and is hugely used in commercial and critical systems. Secondly, Oracle Database uses a different approach for the locking, so the general comparison of the locking is impossible and infeasible, while Oracle does not use *Shared* locks, instead, transaction locks are used to serve the historical consistent data image. Thirdly, it is now made available through the Oracle Cloud Infrastructure [7] by allowing to provision of autonomous databases [1] [2], which are maintained, secured, and patched by the cloud vendor. Fourthly, to make the decisions and information technology correct and complex, proper data analysis must be done. Oracle allows you to create an autonomous data warehouse operated in the cloud environment, supervised by auto-indexing improving performance [7]. Finally, this paper is supported by the Erasmus+ project EverGreen [16], in which Oracle acts as a consortium partner. Efficient data and lock management is critical in application development. Rapid data-driven applications can be easily done by the Oracle APEX technology. Erasmus+ project BeeAPEX [15] emphasizes such a development using low-code programming.

Spatio-temporal model holding airplane locations and Flight Information Regions (FIR) in aviation assignment was used for the computational study and performance evaluation. Each plane was regularly periodically monitored by focusing on the flight parameters. Temporal attributes refer to the entry and exit time of the aircraft from the particular FIR area. The data set consisted of 5 million records in the European region during 2017 – 2020. The example of the data layer is shown in Fig. 1.

```
"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"
```

Fig. 1. Data source structure

This paper aims to deal with the *Reservable* attribute definition, navigating the system to use lock-free

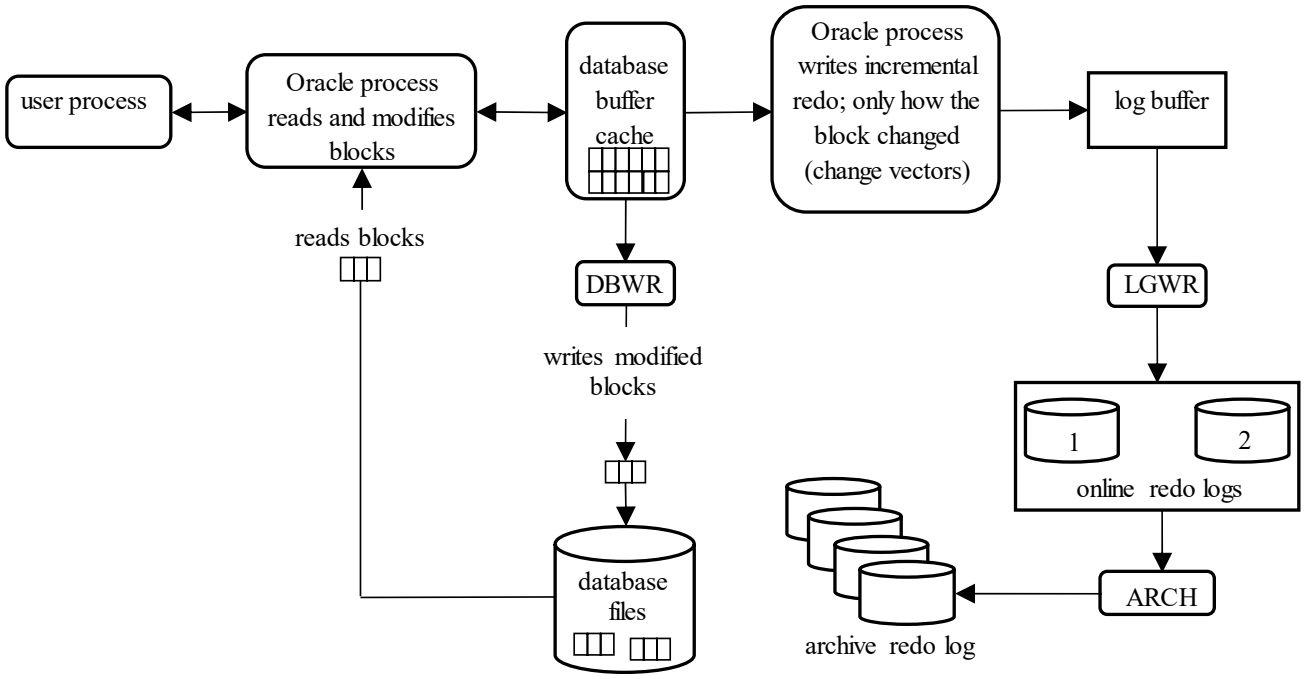

Fig. 2. Transaction logs [7]

update operations for that. Besides, it evaluates a *Reservation journal* [18] data structure by focusing on the performance, compared to the existing data warehouse approaches to serve as a workaround to limit contention.

For the purposes of the computational performance study of the data locking enhancements referrable by the Oracle Database, this paper is structured as follows. Section 2 deals with the transactions and ACID properties. Section 3 emphasizes integrity and deferrable constraints by shifting the consistency check to the end of the transaction. Section 4 deals with the locking. Reservable attributes are displayed and discussed in the section 5. The performance evaluation is treated by section 6 by referring to the data warehouse and function-based index.

II. TRANSACTIONS & ACID PROPERTIES

The main aspect of the relational paradigm, except for the data structure itself, is the transaction support ensuring, that all integrity constraints and rules are passed not later than at the end of the transaction. Thus, the transaction shifts the database from one consistent state to another, which is also consistent. Therefore, it is impossible to load data, which do not cover the rules of the data model, application domain, and user definitions completely. There are four properties of the transactions with the acronym ACID – *atomicity*, *consistency*, *isolation,* and *durability*. *Atomicity* ensures that the transaction is done either completely or it is completely refused. Thus, it is impossible to accept only part of the transaction, like some operations would be accepted, and the rest would be refused. *Consistency* ensures, that all the constraints must be passed before approving the transaction and making the changes durable. From the data locks point of view, the next two properties are critical. *Isolation* supplies the change operation spread after reaching commit. Thus, during the transaction, individual operations and changes are done only locally and other sessions or transactions do not have access to the particular data. This property is related to the building consistent data image throughout the run by using transaction logs and change vectors. Finally, durability ensures, that the approved transaction cannot be lost, even after the instance collapses. It is supervised by the transaction log journals, as well [5], [7], [10].

Fig. 2 shows the structure of the transaction log management in Oracle Database. The main part is delimited by the online logs, which store the change vectors and transaction reference for the active transactions or data, which can be necessary to obtain historical data consistent images for other active transactions. There are multiple logs, which are formed in the circular linked list. Thus, data about the transactions, that are finished, can be consecutively removed. In general, logs are defined by groups consisting of multiple mirrors to ensure robustness and failure resistance. Among that, before the online log rewrite operation, the particular block can be copied to the archive repository. By using them, it is possible to reconstruct the database to any state to limit the failure. Furthermore, by applying online and archive logs, any data image valid in the past can be built. Fig. 1 also shows the data flow and change operations, preceded by the data block loading into the memory (*Database Buffer Cache*). The writing operation from the memory into the physical database is operated by the *Database Writer* (*DBWR*) background process. The active online transaction logs are managed and modified by the *Log Writer* (*LGWR*) background process, while archiving is handled by the *Archiver* (*ARCH*) process.

The next section deals with integrity as a critical part of the consistency management rules.

III. INTEGRITY

Integrity is a set of rules applied on various levels, from the data model up to user considerations and application domains. There are five types of integrity rules. *Column* integrity deals with the primary key candidates by focusing on the unique and

duplicate values, as well as the possibility to hold undefined values for the set that can be cored by the column integrity. *Domain* integrity focuses on the data types and applicable values for the individual attributes. Primary keys are covered by the *entity* integrity, while foreign keys refer to the *referential* integrity type. All other types are covered by the *user* integrity. Most of these types are checked immediately during the execution of the particular operation. However, referential integrity can be postponed to the end of the transaction, mostly due to the relationship cycles in the data model or NOT NULL self-relationships. It is done by the deferrable constraints navigating the transaction manager to check the referential integrity not immediately, but at the end of the transaction [3], [5].

A. Deferrable constraints

Constraints marked as deferred are not checked until the transaction is to be committed. Deferrable constraints are listed for each foreign key constraint separately and applied on the constraint level. The keyword *Immediate* forces the system to check the constraint immediately during the operation, while *Deferred* allows to shift the check processing till the end of the transaction. The syntax for the constraint definition is following:

```
alter table <table_name>
 add [constraint <constraint_name>]
   foreign key <list_of_attributes>
     references <table_name>
                [(<list_of_attributes>)]
     [{initially immediate | deferrable}];
```

In addition to the foreign key, the deferrable constraint can also be applied to the unique constraint, like:

```
alter table flight add flight_no integer
                primary key deferrable;
```

or

```
alter table flight add
  unique(airline, flight_reg_no) deferrable;
```

Immediate option is used by default.

To allow the shift of the checking, session settings must be also set:

```
alter session
  set constraints = [{immediate | deferred}];
```

In the huge parallel processing environment, proper integrity checks, timing, and locking are critical from the performance perspective. In the section section, transaction lock types are defined.

IV. Transaction Locks

SQL norm defines two lock types to be applied. *Shared* locks are associated with the data access and reading operation. Multiple *Shared* locks can be applied simultaneously, meaning, that multiple transactions can access the same data portion in parallel. *Exclusive* locks are defined for the data change operation and are exclusive, meaning, that only one transaction can modify the row [4], [5], [7], [12]. The rest ones are put to the access change list and must wait. That is the definition of the SQL norm. The lock map is shown in Table I.

TABLE I. Lock map

Required → Existing ↓	Exclusive	Shared	No lock
Exclusive	×	×	✓
Shared	×	✓	✓
No lock	✓	✓	✓

However, Oracle Database uses another approach. Transaction logs are hugely used to compose data image as it existed at the beginning of the transaction or executed operation. Therefore, it is a stronger pressure to make the logs available as long as possible, naturally, based on the workload and online log capacity. If it is impossible to get a consistent data image for the processing and execution, *Snapshot too old* exception (*ORA-01555*) is raised. Limiting *Shared* locks offers a significantly wider spectrum of parallel data access and processing. Moreover, even after the change operation of the row, which is to be accessed by another transaction, no waiting is necessary, original data values are obtained from the logs. Consequently, Oracle Database uses only *Exclusive* locks for the data load and change operations. The applied granularity is the whole row, except for a structural change of the table, in which the entire table is locked. This can happen naturally only after releasing all *Exclusive* locks on records. *Shared* logs are not used, at all.

However, what about the releasing of the locks? Well, naturally, they must be released not later than the end of the transaction. Simply, it is impossible to share locks across multiple transactions.

There are several techniques for locking and unlocking to serve the workload, spread the opportunity for parallel processing, and limit negative aspects, like deadlocks, waiting, etc.

It is, however, rather a logical concept of the lock release than physical. Namely, the locks are part of the data block and physical release would require to access the data blocks multiple times. Furthermore, the block does not need to be in the instance memory, so the additional I/O operations for the loading would be necessary. To ensure correctness, when accessing the data block, Oracle RDBMS automatically checks the applied locks and release expired. The list of active transactions, which can lock the data are still available for the database manager and background processes of the instance. Therefore, expired locks can be easily identified, while each lock is associated with the base transaction.

It is evident, that the data locks applied in the transaction are an inevitable part of ensuring data correctness and reliability in case of attempting to change the data row by multiple transactions in parallel. However, there are situations, when data change operation can be placed, but no locks are applied. In the next section, the *Reservable* keyword associated with the attributes, is introduced.

V. RESERVABLE ATTRIBUTES

A. Practical issue

The problem with the concurrency can be easily interpreted by getting the current amount of products in-stock, number of sold items or by getting the number of accidents in a region. To point to the issue, we aim to get the number of airplanes in a specified FIR. Naturally, it can be calculated from the data source by taking into account the time of entry and exit from a particular region. However, in critical systems, such information should be obtained very fast to serve the security and monitoring ability. Thus, instead of dynamic calculations on demand, each region is characterized by an attribute holding the current value of planes covered by it. Thus, it would be necessary to pick the value and increment or decrement it, based on the current situation. However, it should be done inside the transactions, requiring the database manager to lock the row. Practically, it would mean locking the whole FIR and release it after the processing. Consequently, it would be impossible to enter or exit the region at the same time. Moreover, one must strongly understand the meaning of the word at the same time. It is not just about the physical time but the reference to the transaction itself during which the FIR is blocked. If we take into account possible communication breakdowns, short-term loss of signal, or air corridors with a large amount of traffic in them, the problem takes on huge dimensions.

Practically, it was impossible to be solved in an efficient manner. Namely, there were several solutions to be used. In the first solution, the current value (number of airplanes in a region) was not stored. Instead, the value was calculated dynamically, which required a large amount of data to process and delays. The second approach used a materialized view, which was, however, refreshed only periodically by the defined frequency, so the provided value was not warranted. The third solution was commonly based on the NoSQL solutions limiting the transactions, so the locking was not used. This approach, however, led to many inconsistencies, because the value had to be loaded first and then incremented, or decremented. In principle, however, several transactions could load the value, and as long as they changed it, it could already be processed by another transaction. As a result, an aircraft entering the region was not recorded, or conversely, if it had already left the territory, it was still considered to be present in the area. The problem is shown in Fig. 3.

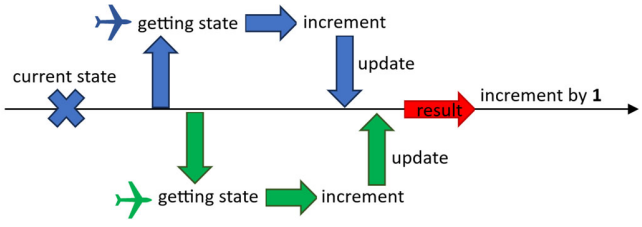

Fig. 3. Lost update

Locking of the FIR does not make sense, as it generates many waiting periods losing precision and accuracy (Fig. 4).

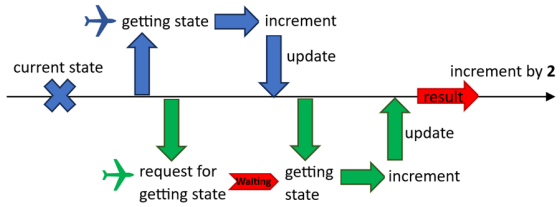

Fig. 4. FIR locking

Autonomous transactions partially solved the problem by excluding the number of associated airplane management into a separate transaction, which was significantly shorter, so the locking was minimized. One way or another, it would be still necessary to synchronize outer and inner transactions, while the outer transaction (monitoring the whole flight) must always end in the same manner as the region monitoring (inner transaction).

Reservable attributes bring a simplified solution by providing inner transaction management automatically.

B. Solution using Reservable attributes

Oracle Database 23c introduces lock-free attribute change aiming to reduce contention of multiple transactions [17] [18]. In real systems, in which the huge parallelism is used, it would be so critical. Lock-free attributes can be specified for frequently updated attributes, defined by the numerical domain. It does not allow data corruption, because the value management is still covered by the transaction, however, the real data value change is done at the end of the transaction, immediately before performing transaction approval. Physically, it takes the current value, makes a change operation and immediately ends the transaction by releasing the locks completely. Thus, the high parallelism aspect is ensured. The data flow and lock management are shown in Fig. 5.

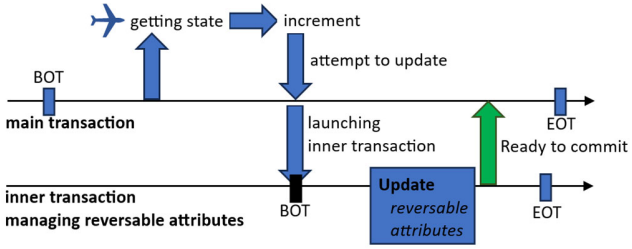

Fig. 5. Reservable attribute management

The definition of the *Reservable* attribute is following:

```
create table fir_tab
    (fir_id integer primary key,
     cur_state integer reservable,
     inbound integer reservable,
     outbound integer reservable);
```

For each *Reservable* attribute, a specific table holding the attempts for the update is created – *Reservation table*. It can be referenced by the data dictionary view referring to the *{USER|ALL|DBA}_OBJECTS* structure.

```
select object_name
 from user_objects
  where object_type= 'TABLE'
   and object_name like 'SYS\_RESERV%'
                         escape '\';
```

The name of the table is system generated, prefixed by the *'SYS_RESERV'*. It is associated with the user-owned table, so it is visible through the *USER* data dictionary view category. It is owned by the same user as the owner of the table:

The *Reservation table* consists of multiple attributes by referring to the source table, source attribute, executed operation and value to be processed. The names for the attributes are partially inherited from the source table:

```
desc SYS_RESERVJRNL_83339
Name            Null?    Type
--------------- -------- ------------
ORA_SAGA_ID$             RAW(16)
ORA_TXN_ID$              RAW(8)
ORA_STATUS$              CHAR(12)
ORA_STMT_TYPE$           CHAR(16)
FIR_ID          NOT NULL NUMBER(38)
CUR_STATE_OP             CHAR(7)
CUR_STATE_RESERVED       NUMBER(38)
INBOUND_OP               CHAR(7)
INBOUND_RESERVED         NUMBER(38)
OUTBOUND_OP              CHAR(7)
OUTBOUND_RESERVED        NUMBER(38)
```

The *Reservable* attribute is set using the defined trigger, based on a particular FIR identifier and the exit_time from the region:

```
create or replace trigger trig_cur_state
 before insert on flight_positions
  for each row
   declare val integer;
    begin
     case when :new.exit_time is not null
               then val:=-1;
          else val:=1;
     end case;
      update fir_tab
       set cur_state=cur_state + val
        where fir_id=:new.fir_id;
   end;
   /
```

If any airplane entrances the FIR, the current number of assigned (*CUR_STATE*) airplanes to the particular FIR is incremented using the trigger. Vice versa, if the *EXIT_TIME* from the FIR holds the real defined value, the particular *CUR_STATE* value is decremented by one.

So, let´s check the solution practically. If the new airplane is to be assigned to the FIR, a trigger is fired and the update request is recorded in the reservable table list, not the attribute value itself. Therefore, by requesting the value of the *CUR_STATE* attribute, the original value is obtained!

```
select * from fir_tab;

FIR_ID  CUR_STATE    INBOUND   OUTBOUND
-------- ----------- -------- ------------
2        5           8         3
```

At this moment, the *Reservation table* content looks like following:

```
select ORA_STMT_TYPE$, FIR_ID,
         CUR_STATE_OP, CUR_STATE_RESERVED
 from SYS_RESERVJRNL_83339;

ORA_STMT_TYPE$             FIR_ID
     CUR_STA       CUR_STATE_RESERVED
---------------- ---------- ------- ------
UPDATE                     2
     +             1
UPDATE                     2
     +             1
UPDATE                     2
     +             -1
```

By reaching the transaction end (*Commit*), the *Reservation* table for the referenced transaction is freed by applying the changes physically for the attribute:

```
commit;

select * from fir_tab;

FIR_ID  CUR_STATE    INBOUND   OUTBOUND
-------- ----------- -------- ------------
2        6           9         3

select * from SYS_RESERVJRNL_83339;

     -- no rows selected
```

C. Properties of the Reservable attributes

Reservable attributes require additional data storage capacity and reference in the system tables. The internal management is automated, so no specific user intervention is necessary. The only thing is to mark the attribute to be *Reservable*. The keyword *Reservable* can be applied to multiple attributes at once.

```
alter table products
   modify(in_stock, sold_number reservable);
```

For one table, irrespective of the number of Reservable attributes, one table is created, which can form the limitation

because it is not always necessary to change all Reservable columns at the same time. A suitable alternative could be partitioning of the *Reservable table* based on the marked columns, which, however, cannot be done.

Besides the definition of the Reservable attribute, it is automatically enhanced by the column integrity constraint – NOT NULL, while undefined values cannot be mathematically processed and operated. Thus, it is always necessary to hold there a real value, to which increment or decrement based on the specified value can be done.

D. Reservable attributes remarks

Concluding the *Reservable* attribute definition, there are some rules, which must be used to ensure proper management and functionality [17] [18]:

- Table, which holds a *Reservable* attribute must have a primary key to be able to reference the row in the Reservation table:

  ORA-55728: Reservable column property can only be specified for a column on a table that has a primary key.

- Table, which holds a *Reservable* attribute must have a primary key to be able to reference the row in the Reservation table:

  ORA-55764: Cannot DROP or MOVE tables with reservable columns. First run "ALTER TABLE <table_name> MODIFY (<reservable_column_name> NOT RESERVABLE)" and then DROP or MOVE the table.

- The only available operations for the Reservable attribute are (+) and (-):

  ORA-55746: Reservable column update statement only supports + or – operations on a reservable column.

- The row, which is currently enhanced by the active records in the *Reservation* table cannot be removed before ending the blocking transaction:

  ORA-55754: Resource busy error is detected for the reservable column update statement. A delete or a DDL operation is conflicting with this update statement.

VI. PERFORMANCE STUDY

For the performance evaluation and data management study, a flight monitoring data set was used, consisting of 5 million of records defined by the airplane positions, flight parameters and FIR references. For each FIR region, the current number of assigned airplanes was stored. To sharpen the solution, there was also evidence of the total number of inbound and outbound flights. Please note, that the FIR definition can evolve over time and is not static. In general, it does not reflect the borders of the countries. FIR in Europe is supervised by the Eurocontrol. Fig. 6 shows the FIR assignment segment in Europe.

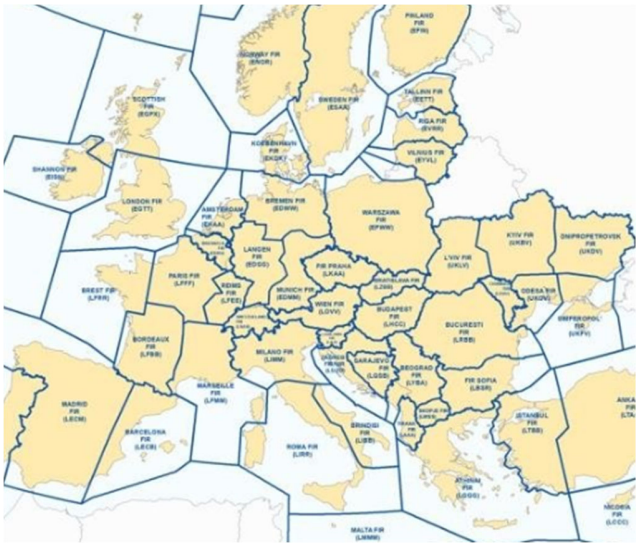

Fig. 6. European FIRs [13]

The first part of the study deals with the impact of *Reservable* attributes management. The defined *fir_table* (50 rows) contains 3 *Reservable* attributes – *cur_state*, *inbound* and *outbound*. The value is set by the trigger, based on the *flight_positions* table (loading 5 million rows). For the purposes of the evaluation, only FIR entrance and exit points are taken, positions inside the FIR, which do not reflect *Reservable* column values, are not taken into consideration. All the data are processed in one active transaction and overall processing time is evaluated. The results are shown in Table II Ordinary columns do not consider *Reservable* columns, however, in a practical environment, it would generate multiple waiting operations caused by the locking. The second solution emphasizes *Reservable* columns by using lock-free solution by placing the change requests in the *Reservable* table.

TABLE II. RESULTS – TRANSACTION WORKING PHASE

	Ordinary columns	**Reservable columns**	**Real environment**
	3 monitored attributes		
Data loading *[INTERVAL DAY(9) TO SECOND(3)]*	79:42.195	608:25.648	1247:70.403

While the first experiment focuses on the ***working phase of the transaction***, the second evaluation experiment deals with the ***certification phase*** by monitoring the process of applying changes temporarily stored in the *Reservation* table, followed by freeing it. If the ordinary columns are used, only active online locks are copied into the database and transactions can be flagged as successfully ended. However, if the *Reservation* table is used, the whole process contains an additional step – applying changes to the data, which must precede the transaction approval. In this part, therefore, additional processing time demands are taken into consideration. The reached results are shown in Table III. Each evaluation run was performed 100

times, the results express the average value to limit any side server impacts.

TABLE III. RESULTS – TRANSACTION CERTIFICATION PHASE

	Ordinary columns	Reservable columns	Real environment
	3 monitored attributes		
Data aproaving *[INTERVAL DAY(9) TO SECOND(3)]*	2:46.381	13:26.735	2:46.381

Reservable attribute management requires significant additional resources in terms of processing time, as well as the storage capacity. When dealing with the transactions by splitting them into working and certification phases, performance impacts can be complexly evaluated. To load 5 million rows consisting of the flight positions and parameters, total demands are 79.42 seconds, which expresses only 13.06%, compared to the *Reservable* columns. In other words, *Reservable* column management required additional 528.83 seconds, which represents an increase of more than 655.9%. Similarly, the certification phase of the *Reservable* column management requires update statements execution. Additional processing time demands of this phase are 10.8 seconds (439.02%). At first sight, it can be considered as a huge increase and the solution should be refused. However, this reflects only an ideal solution with no parallelism and waiting times. Thus, by using a practical real environment, the total demands of the transaction working phase are 1247.70 seconds. By applying *Reservable* columns, processing time demands are lowered to 608.25 seconds, which reflects the drop using 51.25%. Even the certification phase is more demanding, overall, the total processing time cost drop reflects 50.29%. The reached results in a graphical chart form are in Fig. 7.

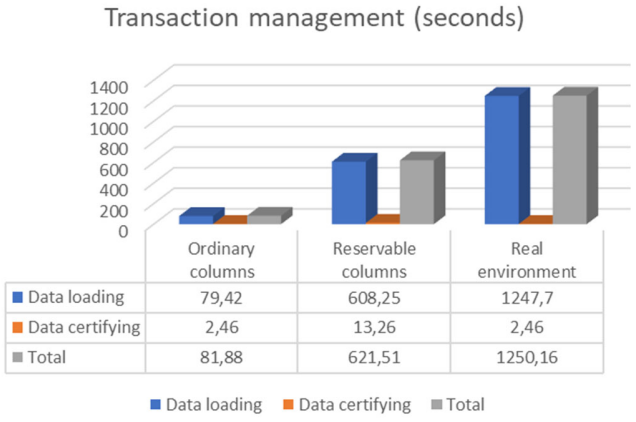

Fig. 7. Transaction management processing time results

Finally, the *Reservation* table size is considered, which required 2 GB for 5 million changed rows with 3 *Reservable* columns. The limitation of the approach is related to the vacuuming and freeding of the table. Whereas the *Reservation* table is treated as a standard table with the segment and extents, the already allocated storage is released by the transaction end. Although the blocks can be later used, the required storage does not reflect the current usage, but the highest historical peak. In the environment of the parallelism and flight monitoring, it practically reflects the most significant extraordinary events, mostly represented by the request to free the airspace and airspace closure.

VII. CONCLUSIONS

Reservable column management provides a powerful solution in the massive parallelism, which requires increment or decrement of the particular attribute set, based on the conditions. To serve the workload and minimize waiting time, lock-free solution was proposed in the Oracle Database 23c release. Instead of the direct update operation by applying Exclusive transaction locks limiting any other parallel access, the request is recorded in the system-managed *Reservation* table, by referring to the attribute, value to be added or removed. The merge operation and applying the change for the *Reservable* columns physically is shifted to the end of the transaction – at the beginning of the certification phase of the transaction. Thus, the locks are applied for only time intervals, just to ensure the loading and changing value in a single unit by applying the atomicity property of the transaction.

As stated, it does not make sense to specify *Reservable* column for non-parallel processing or for the systems, where the waiting for the lock release is not critical. However, for the analytical processing, and system monitoring (medical treatment, transport systems or e-shop), accessing proper values immediately is required.

This paper aims to provide a performance study of the *Reservable* column management introduced in the Oracle Database 23c release. In future research, we will focus on creating our own solution for the lock-free Reservable column management, enhanced by the triggers and autonomous transactions managing data in the *Reservation* table. However, the assumed structure is not a flat table, but it will be automatically partitioned based on the transaction reference. It would allow to purge (truncate) the partition dynamically if the transaction is ended. Thanks to that, associated blocks could be deallocated, mostly in case of a huge transaction. Moreover, there would be an option to specify priorities across multiple *Reservable* columns to optimize the performance.

Another aspect of the future research relates to the data distribution, synchronization across the nodes, and efficiency of the energy consumption and management to sharpen the Green computing paradigm.

ACKNOWLEDGMENT

This paper study was supported by the Erasmus+ projects:

- Project number: 2022-1-SK01-KA220-HED-000089149, Project title: Including EVERyone in GREEN Data Analysis (EVERGREEN) funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the Slovak Academic Association for International Cooperation (SAAIC). Neither the European Union nor SAAIC can be held responsible for them.

Co-funded by the European Union

EverGreen DATA Analytics

- Project number: 2021-1-SI01-KA220-HED-000032218, Project title: Better Employability for Everyone with APEX.

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

BEE WITH APEX

REFERENCES

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

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

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

[4] Greenwald, R., Stackowiak R., and Stern, J.: *Oracle Essentials: Oracle Database 12c*, O'Reilly Media, 2013.

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

[6] Kvet, M.: 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

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

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

[9] Limaye, N. et all.: Thwarting All Logic Locking Attacks: Dishonest Oracle With Truly Random Logic Locking, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (Volume: 40, Issue: 9, September 2021

[10] Morris, S.: Resilient Oracle PL/SQL, O´Reolly, 2023.

[11] Nuijten, A. and Barel A.: Modern Oracle Database Programming: Level Up Your Skill Set to Oracle's Latest and Most Powerful Features in SQL, PL/SQL, and JSON, Apress, 2023

[12] Rosenzweig, B. and Rakhimov, E.: Oracle PL/SQL by Example, Oracle Press, 2023.

[13] Standfuss, T. and Schultz, M.: Performance Assessment of European Air Navigation Service Providers, DASC conference 2018

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

[15] Erasmus+ project BeeAPEX - Better Employability for everyone with APEX : https://beeapex.eu/

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

[17] Lock-Free Reservations to Prevent Blocking Sessions in Oracle Database 23c: https://oracle-base.com/articles/23c/lock-free-reservations-23c

[18] Using Lock-Free Reservation: https://docs.oracle.com/en/database/oracle/oracle-database/23/adfns/using-lock-free-reservation.html#GUID-60D87F8F-AD9B-40A6-BB3C-193FFF0E60BB