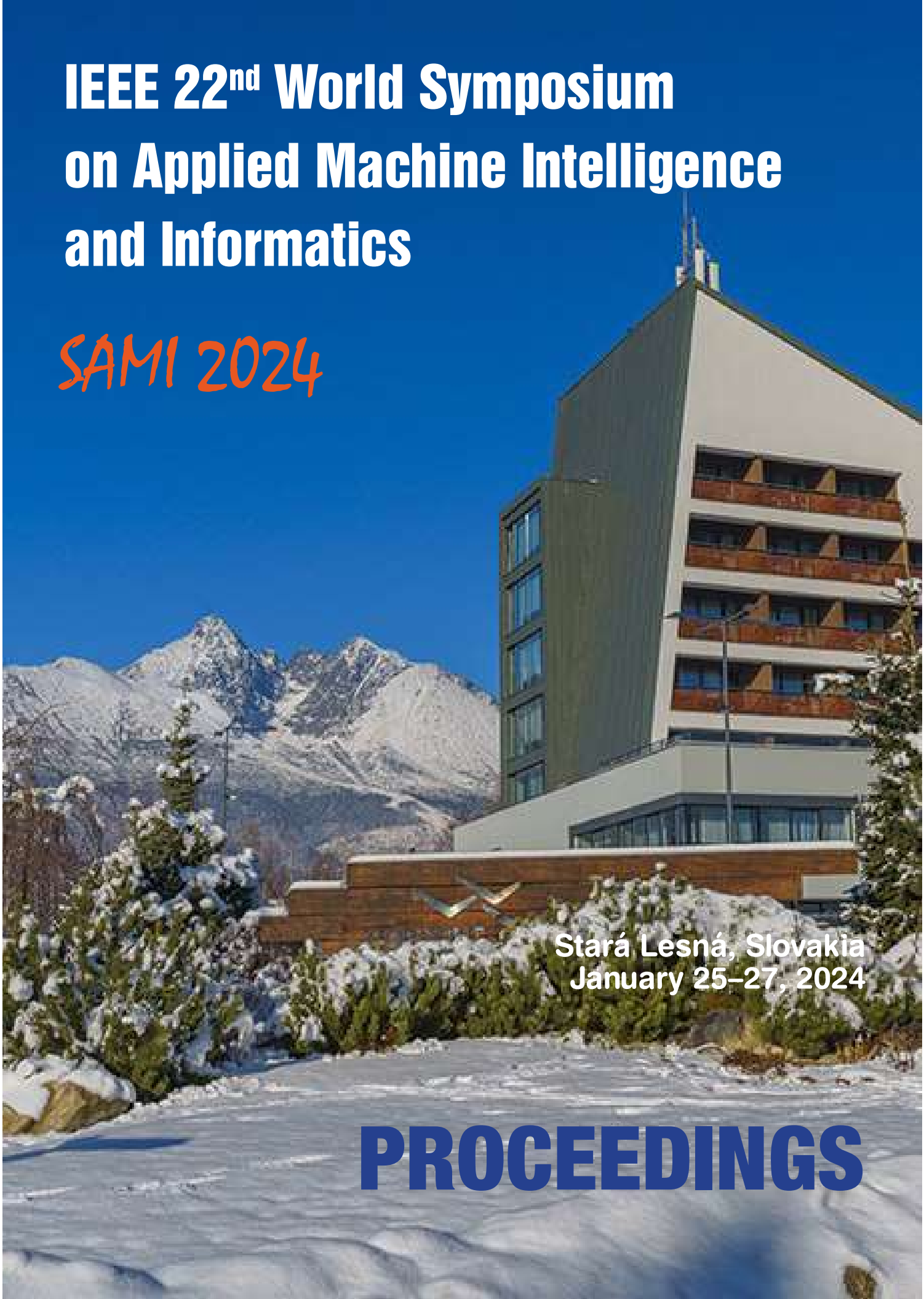
IEEE 22nd World Symposium on Applied Machine Intelligence and Informatics

SAMI 2024



Stará Lesná, Slovakia
January 25–27, 2024

PROCEEDINGS

Organizers and Sponsors

Technical University of Košice, Slovakia
Óbuda University, Budapest, Hungary
University Research and Innovation Center
Antal Bejczy Center for Intelligent Robotics
ELFA Ltd., Košice, Slovakia
Slovak Academy of Sciences
Hungarian Fuzzy Association
SMC TC on Computational Cybernetics
IEEE Computational Intelligence Chapter of Czechoslovakia Section

Sponsors

IEEE Hungary Section
IEEE Joint Chapter of IES and RAS, Hungary
IEEE Control Systems Chapter, Hungary
IEEE SMC Chapter, Hungary

Technical Co-Sponsor

IEEE SMC Society

	Part Number	ISBN
XPLORE COMPLIANT:	CFP2408E-ART	979-8-3503-1720-6
USB:	CFP2408E-USB	979-8-3503-1719-0

Welcome from the Chairs

Computational Intelligence and Intelligent Technologies are very important tools in building intelligent systems with various degree of autonomous behavior. These groups of tools support such features as ability to learn and adaptability of the intelligent systems in various types of environments and situations. The current and future Information Society is expecting to be implemented with the framework of the Ambient Intelligence (AmI) approach into technologies and everyday life. These accomplishments provide the wide range of application potentials for Machine Intelligence tools to support the AmI concept implementation. The number of studies indicates that this approach is inevitable and will play essential and central role in the development of Information Society in close future.

The essential importance of the Machine Intelligence in this historically challenging effort points out the responsibility of MI community including all fields like Brian-like research and applications, fuzzy logic, neural networks, evolutionary computation, multi-agent systems, artificial life, Expert Systems, Symbolic approaches based on logic reasoning, Knowledge discovery, mining, replication and many other related fields supporting the development and creation of the Intelligent System. The importance embedding these systems in various kinds of technologies should bring profit and different role of mankind in production and in everyday life. We expect to have intelligent technologies, solution and even humanoid robots to help the mankind to improve and keep the ideas of humanity and democracy.

The role of Machine Intelligence Quotient will play an important role in the future to be able to evaluate the degree of the autonomous behavior of the designed system. It is belief that it will be domain oriented problem and should also be important to use this information for decisions made by humans e.g. in evaluation of many information system in commercial tender to pick up the system with the highest MIQ. The usefulness of this parameter will be dependent on many influences including technological, domain oriented and also commercial aspects of the CI application in various systems. The commercial need to have “intelligent” solution and products should increase the interest for MI tools.

This year number of contribution showed up from mechanical Engineering domain, control and also pure computer science. We do believe that this multidisciplinarity will be very useful to emerge more AI applications in Information Society and will help making products and solutions more “intelligent”.

This proceedings is a small contribution of knowledge dissemination and presentation of important problems and advances in Computational intelligence theory and applications. Hungary and Slovakia as members of EU will do their best to contribute to European Research Area and support the development of Computational Intelligence technology for the benefit of the mankind.

Levente Kovács and **Liberios Vokorokos**
General Chairs

Committees

General Chairs

Levente Kovács, Óbuda University, Budapest, Hungary
Liberios Vokorokos, Technical University of Košice, Slovakia

Founding Honorary Chair

Imre J. Rudas, Óbuda University, Budapest, Hungary

Honorary Committee

Stanislav Kmet, Technical University of Košice, Slovakia
Anton Čižmár, Technical University of Košice, Slovakia
Levente Kovács, Óbuda University, Budapest, Hungary
Peter Mésároš, Technical University of Košice, Slovakia

International Scientific Committee

Alin Albu-Schaeffer, German Aerospace Center, Germany
Philip Chen, University of Macau, Macau
Paolo Dario, Scuola Superiore Sant'Anna, Italy
Paolo Fiorini, University of Verona, Italy
Hamido Fujita, Iwate Prefectural University, Japan
Huijun Gao, Harbin Institute of Technology, China
Tamás Haidegger, Óbuda University, Budapest, Hungary
Keith Heipel, University of Waterloo, Canada
Oussama Khatib, Stanford University, USA
Kazuhiro Kosuge, Tohoku University, Japan
Gernot Kronreif, ACMIT Gmbh, Austria
Bernd Liepert, KUKA Roboter AG, Germany
Ren Luo, National Taiwan University, Taiwan
Vincenzo Piuri, Universita' degli Studi di Milano, Italy
Bruno Siciliano, University of Naples, Italy
Peter Sinčák, Technical University of Košice, Slovakia
Masayoshi Tomizuka, University of California, Berkeley, USA
Jacek Zurada, University of Louisville, USA

International Organizing Committee Co-Chairs

Frantisek Babič, Technical University of Košice, Slovakia
Marián Bucko, Elfa, Slovakia
Ladislav Főző, Technical University of Košice, Slovakia
Norbert Ádám, Technical University of Košice, Slovakia

Technical Program Committee Chairs

Szilveszter Kovács, University of Miskolc, Hungary
Rudolf Andoga, Technical University of Košice, Slovakia
Ivana Budinska, Slovak Academy of Science, Slovakia

Technical Program Committee

Norbert Ádám, Technical University of Košice, Slovakia
Rudolf Andoga, Technical University of Košice, Slovakia
František Babič, Technical University of Kosice, Slovakia
Péter Baranyi, Széchenyi István University, Győr, Hungary
Peter Bednár, Technical University of Kosice, Slovakia

Secretary General

Anikó Szakál
Óbuda University, Budapest, Hungary
E-mail: szakal@uni-obuda.hu

Iveta Zamecnikova
Technical University of Košice, Slovakia
E-mail: zamecnikova@elfa.sk

Table of Contents

Top data analysis performance – case study

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

***Abstract*—Obtaining top data based on defined criteria requires sorting the data, which can be applied to all data in one group or to individual data partitions specified in analytical functions. The limitation of the database system performance is associated with the cardinality of the data set processed in the individual evaluation steps, especially regarding data warehouses and large sets that need to be joined. However, most of the data are refused while maintaining only top data. Many solutions are available, but the significant aspect is related to the performance from the system resources and time consumption point of view. This paper aims to analyze individual solutions and techniques and provide a methodology for processing top data not only in the data warehouse but also in an online transaction processing environment limiting the index number range.**

Keywords—Data analytics, Oracle Database, Performance, Transaction database, Indexing, Warehouses

I. INTRODUCTION

The data amount to be handled is still rising, shifting the processing from the conventional sphere dealing with only current valid data to the temporal environment, allowing monitoring of the data evolution in the whole time spectrum [1]. Several metrics categorize the data based on the origin, format, applicability, reliability, as well as precision frame and time reference [1] [4] [5] [15]. Temporal databases are continuously developed and improved, starting from the object level architecture enhancing the original object identifier by the validity time frames through the attribute-oriented granularity encapsulating each data column with the time reference, up to synchronization groups, which emphasize data value as a core granularity [3]. Data value can reflect either the attribute itself or a set of attributes, which are temporally treated as one segment. Thanks to that, disc storage demands and the process of obtaining object state at the defined timestamp is done easier and faster. Temporal architectures form the interlayer between conventional relational data management and data warehouses, offering the layer for data analytics. Currently, complex data sets to be analyzed can originate from various sources and can be stored in any database system and architecture [2] [11] [12].

Data to be analyzed can be stored in the online transaction processing system, characterized by the short transactions getting new data or modifying existing. In the temporal environment, the update operation is rather logical, while it always forces the system to create a new state and persist the whole evolution. However, there can also be requirements to correct already stored and loaded data by introducing multiple temporal spheres – validity, transaction reference, up to IPLT models delimited by multiple timestamps for each data node – input stream (adding a record to the input queue for processing), processing timestamp, load timestamp, and transaction completion time record. All these temporal references are essential in real-time processing systems, where any delay can be critical [9].

In addition, the data to be analyzed, can be stored in extensive data structures, mainly in data warehouses, mostly determined by the star schema, marts, or their subvariants and extensions [12] [13] [16]. It is usually necessary to obtain the most critical data, the most significant changes, the fastest time, and simply top data based on defined criteria. Thus, the data need to be sorted and then evaluated. The process is relatively straightforward if the index for those criteria is present. Simply, leaf nodes are treated, and top data are obtained and processed. However, if there is no suitable index and the data set is critically huge, the performance can degrade. Furthermore, if the original data set must be joined to other tables, cardinality can significantly rise. On the other hand, it should be kept in mind that we still only need to obtain the top data, and thus, most of the originally sorted data do not need to be longer processed.

The proposed paper aims to analyze the performance of the queries getting top data regardless of the data storage, architecture, and infrastructure, operated by the SQL language. It uses three data sets with various cardinality to declare the robustness operated by the scalability option. We do not intend to combine and evaluate multiple database systems. Instead, the focus is on the Oracle Database, which forms the most relevant database structure for complex data analytics in the cloud environment, even enhanced by machine learning techniques to get the prognoses, identify patterns, etc. Furthermore, the Oracle Database is the most powerful and scalable system in terms of accessibility through indexes, data loading and dynamic parameters for the memory structure size. Finally, this paper is supported by the EverGreen project [18] dealing with environmental green data analytics, in which Oracle Corporation acts as an associated partner.

The structure of the paper is as follows. Section 2 deals with the problem definition, escalating the execution plan of the Select statement. Section 3 references indexing strategies and methods for table joining. Section 4 enhances data analytics, top data meaning, and reference, followed by the individual analytic-oriented clauses and window frames. The evaluation techniques and strategy are discussed in section 5. Performance study is presented in section 6, forming the results and methodology for getting top data in an performance-efficient way.

II. EXECUTION PLAN

SQL is a non-procedural language in which user just specifies what data should be provided and output format, but the internal process of data access must be selected by the database optimizer and transaction manager. Query optimization and selecting an access path is one of the most critical operations, and it strongly influences the performance

of the query execution. When dealing with the defined statement, the following steps must be done (Fig. 1) [6] [7] [8]:

- *Parse*. Before parsing itself, the defined SQL statement is transferred from the user process on the client site to the Oracle interface. The parsing process then checks the validity of the statement by verifying the table structures, column definitions, privileges, etc. Parse locks are applied for all used structures to ensure that the definitions are not changed during the operation. The most suitable execution plan is identified based on the table statistics, optimization, and heuristics. The selected execution plan is stored in the shared SQL area using query hash. Thus, later on, if the analogous statement is to be executed, the Parse stage can be omitted and the already pre-defined plan can be directly applied.
- *Query processing* ensures read consistency by using temporary segments. It is also applied for nested queries.
- *Result description* phase is used, if the query is interactively entered by the user and query result types and formats cannot be directly determined.
- *Defining output* by specifying the size, variable data types in PL/SQL to make the system perform relevant implicit data type conversions, if necessary.
- *Binding variables* by replacing placeholders with real values. Using binding queries allows using the same execution plan for multiple queries differing just with the condition values.
- *Executing statement* by ensuring consistency and integrity. The execution and data access.

The selection of the access path is associated with the database statistics describing the insight of the table content, so it is crucial to make them actual. The process of the execution starts with taking the data sources (tables specified in the From clause), applying the conditions in the Where clause, and getting the output structure – projecting the data in the Select clause. Besides, expressions and functions are considered across the whole statement definition. Starving away from aggregation functions and groups definition, which is done after considering the data source, it is important to mention that the Where clause specifying conditions is treated in the first phase. The problem is that the condition taking top data must be considered after sorting the data. Thus, the whole data set must be treated, either defined by the fetch process limiting the output (evaluated after Order by clause) or by using analytical functions, which encapsulate Order by definition directly in its specification. One way or another, all data are processed, joined, and evaluated, and only then, the top data are selected. The following section considers the impact of the table index and join operation methods used by the Oracle Database.

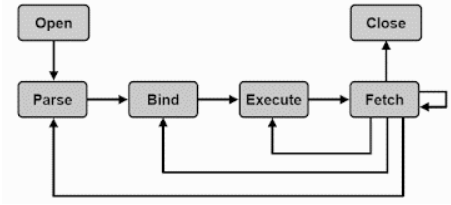


Fig. 1. SQL statement processing phases [7]

III. TABLE INDEX AND JOIN METHODS

An index is an optional structure, which is very effective for data location using the index key. Generally, B+tree index, Bitmap index, and Hash index types are used. Oracle Database no longer support Hash indexes, although its definition can be partially emulated using function indexes.

Typically, B+tree indexes are used in transaction processing. They are formed by the balanced index. The traversing is based on the index key. The leaf node consists of the ROWID pointers to the particular data row residing in the block of the data file in the physical storage, operated by the database layer. Furthermore, the index leaf layer can provide a sorted list of the data based on the index key, which can be formed by the data table attributes, or function results can be used as a key, as well. Index key can be composite, formed by multiple number of attributes or function calls. Primary keys and unique constraints are automatically enhanced by the indexes, but users can also create their own ones to increase the performance. An example of the B+tree is shown in Fig. 2. It is suitable for high cardinality, high degree of distinct columns [7] [17].

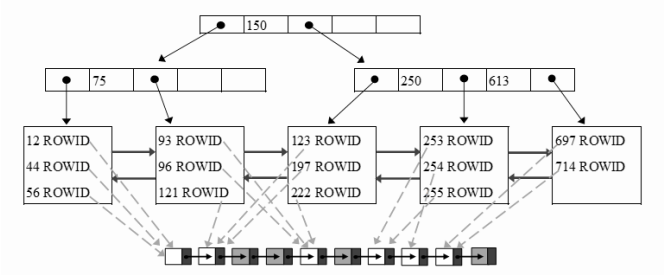


Fig. 2. B+tree

For the data warehouses, marts, and any other analytic-oriented architecture, among B+tree indexes, bitmap indexes are used, characteristic for the low cardinality columns by pointing to the OR and AND operations. Bitmap operations are really fast, so the evaluation and data access can significantly benefit. On the other hand, a bitmap is strongly limited by its structure and any change possibilities in terms of adding new value. Any change in the structure requires rebuilding the whole index. Therefore, it is mostly applicable for static historical data located which are not changed.

Accessing the data using the index can bring a markable improvement since the sequential data block scanning necessity is replaced by direct access using ROWID pointers obtained by the index traverse. Thus, the complexity of the processing is shifted from O(n) to O(log(n)).

A. Join methods

Oracle Database provides three join methods implementing a logical connection between two sets of data. The scenario and method selection depends on the available indexes, the number of estimated rows in each data set, but the most critical part relates to the table statistics and their accuracy [10].

Nested loop is the simplest operation. For each record from the outer input (small table), matching rows are found from the inner set. The joining is based on the primary (unique) key and foreign key. A nested loop is selected in case the foreign key is not indexed. The complexity of the query is O(N * log(M)). M and N express the cardinality of the data sets.

Merge join method is the most efficient way using the fact that both sets of data are indexed based on the joining keys. The complexity is reduced to O(M+N).

Hash match method is an inter solution if the data set to be joined is large, but not enhanced by the index. In that case, a temporary hash index is built to split the data into the buckets, followed by the sequential scanning and mapping, but only across one bucket, which can be done in parallel, but also data amount to be treated is strictly limited. The complexity is O(N * hc + M * hm + C), where N is a smaller data set, hc expresses the complexity of hash index creation, M refers to the larger data set cardinality, hm delimits the complexity of the hash match function. And finally, C addition refers to the complexity of the dynamic calculation and hash function creation.

The following section deals with the data analysis pointing to the top data definition and representation.

IV. Data Analysis – Top Data

Data analysis is the next developmental stage of data aggregation, which is characterized by reducing the output data rows – one for each group defined in the Group by clause. Analytical functions behave differently. Instead of producing one row for each group, the original data set is retained, and each row is extended by the partial analytical function result. By default, for each row, all the preceding rows are considered, but the behavior can be set using the windowing_clause of the analytic function by defining the frame of rows or values that are relevant for processing particular row values [13] [14].

To get the correct results, the data to be handled must be sorted using order_by_clause directly embedded in the analytical function. Thus, instead of considering the whole data set, the sort operation is done directly inside the analytics. Besides, the analytical function can be applied for each partition (specified in the query_partition_clause) separately by building independent solutions across multiple groups. The syntax of the analytical functions is shown in Fig. 3.

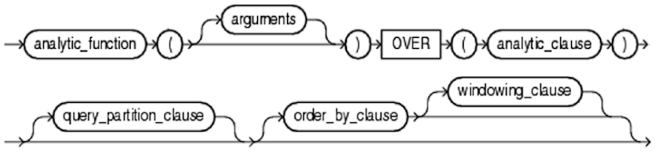


Fig. 3. Analytical function syntax clauses

To get the top data, analytical functions can provide a suitable and easy implementable solution. However, what does the top data actually mean? There are three rules which can be applied:

- **Top-rows rule** getting exactly three rows.
- **Olympic rule** providing gold, silver and bronze the Olympic way.
- **Top-values rule** getting all rows that have top-n values.

The limitation of the top-rows rule is based on the fact that the fourth row is ignored, even though it has the same value as the third one. Moreover, it can be said that the selection of the rows with the same values is random, resulting in the fact that the same query can later provide different results, even based on the same input data. The reason is just based on the Order by clause based on non-unique values.

Tab. 1 shows the function reference for the defined top-rules.

TABLE I. Reference Analytical Functions

Rule	Reference analytical function
Top-rows	Row_number
Olympic rule	Rank
Top-values	Dense_rank

The example for getting top-3 rows is depicted in the next code snippet. Its goal is to get the longest stay of the plane in the flight information region (FIR):

```
select *
 from
  (select flight_id,
          departure_airport, arrival_airport,
          air_company_id,
          row_number() over
              (order by stay_duration desc) as RN
    from flight_data
     where FIR in ('EDGG', 'EDMM', 'EDUU',
                   'EDVV', 'EDWW', 'EDYY')
 )
   where RN<=3;
```

As evident from the snippet, analytical functions cannot be directly placed in the Where clause (referencing FIR regions in Germany), since the order of conditions to be evaluated is not strictly specified and depends on the database optimization process. Besides, not all data are present in the Where clause compared to the final result set. Therefore, any conditions on the analytics must be encapsulated, and a nested query must be created. Specifically, the inner statement calculates the analytical function result for each row. Then, the outer query filters the data based on analytics. The statement execution process is shown in Fig. 4.

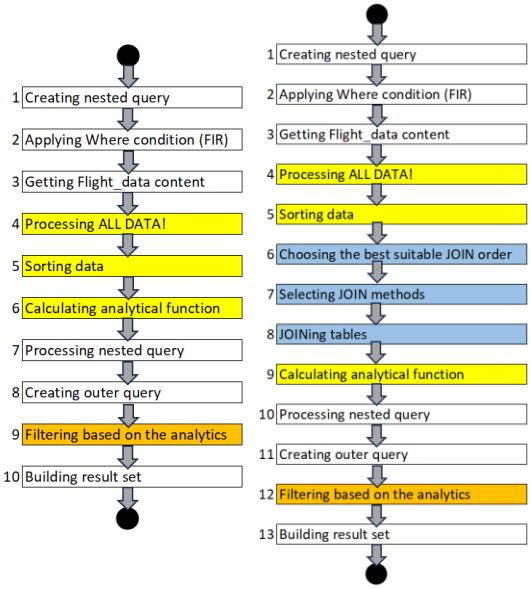


Fig. 4. Statement processing steps – left part - one table, right part – JOIN operation

The first performance limitation relates to the requirement to process all data based on the defined FIR list, whereas the

data need to be sorted and consecutively treated by the analytical functions. However, if multiple tables need to be joined and mapped, the problem is even deeper, whereas the JOIN operation belongs to one of the most demanding database operations. It should be noted that extensive data sets are commonly processed in the analytical environment. The research question is, how to limit the necessity to join all source data together?

V. Strategies for Getting Top Data

For the performance analysis and evaluation, multiple cases were used to declare the processing time demands and costs. The data set monitoring the flights was used, delimited by various cardinalities to declare the scalability of the system:

- **CASE 1:** taking 10 longest flights. For each flight, individual FIR (Flight information region) regions are referenced, enhanced by their parameters at a given time, while they evolve over time. Three data query representations are evaluated:
  - Q11 using analytical function row_number, defined in the inner statement, followed by the filtering based on it in the outer statement.

```
Q11:
select *
 from
 (select fir_parameters(temporal), air_company_data,
         flight_data,
         row_number() over(order by ectrl_id) as rn
   from flight_data join fir using(fir_ref)
                    join air_company using(comp_ref)
 )
   where rn<=10;
```

  - Q12 based on pre-sorting and limiting the output result set number using Fetch first clause.

```
Q12:
select fir_parameters(temporal), air_company_data, flight_data
  from flight_data join fir using(fir_ref)
                   join air_company using(comp_ref)
    order by ectrl_id
     fetch first 10 rows only;
```

  - Q13 postponing JOIN operation to the end of the query processing.

```
Q13:
select fir_parameters(temporal),
       air_company_data, flight_data, rn
from
 (select *
   from
    (select flight_data,
            row_number() over(order by ectrl_id) as rn
       from flight_data
 )
   where rn<=10) inner
      join fir using(fir_ref)
      join air_company using(comp_ref);
```

- **CASE 2:** taking 10 longest flight for each FIR assignment processed independently. This is done by using Partition clause of the analytical function. Following query representations are evaluated:
  - Q21 using analytical function row_number, defined in the inner statement, followed by the filtering based on it in outer statement.

```
Q21:
select *
 from
 (select fir_parameter), air_company_data,
         flight_data,
         row_number() over(partition by fir_list_seq
                           order by ectrl_id) as rn
   from flight_data join fir using(fir_ref)
                    join air_company using(comp_ref)
 )
   where rn<=10;
```

  - Q22 using JOIN LATERAL allowing to dynamically join data on the run.

```
Q22:
select fir.*, inner.*
 from fir + air_company
  cross join lateral
   (select * from flight_data fd
     where fd.fir_list_seq=fir.fir_ref
      order by ectrl_id
       fetch first 3 rows only
   ) inner;
```

  - Fetch first clause cannot be used, whereas it does not work for multiple partitions.

CASE 2 has two variants to be evaluated – partition defined by one attribute (CASE 2a) and partition enhanced by the composite key formed by two attributes (CASE 2b).

For both cases, the sotring rule was based on the table attribute, or the function call was used to focus on the attribute encapsulation in the function call.

VI. Performance

For the computational study, a server with the following parameters was used:

- Operating system: Windows Server 2020
- Processor: AMD Ryzen 5 PRO 5650U with Radeon Graphics, 2.30 GHz
- Memory: 2x 32 GB DDR-4, 3200MHz, CL20
- Disc storage: 2 TB, NVMe, read/write 3500 MB/s

As stated, Oracle Database was used delimited by the version *Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 – Production Version 21.3.0.0.0*. Three data sets were used, differentiated by the cardinality. Two tables were referenced – the FIR table took 100 rows describing the temporal parameters and assignments of the FIR. Each row took 1024 KB. The smallest fragment of the flight data consisted of 10 000 rows. The example of the flight data is depicted in Fig. 5. Each flight is identified by the *ECTRL_ID* and set of obtained data monitoring the flight (*Sequence_number*), current flight parameters, and assignment to the FIR (*AUA_ID*). That assignment is temporally bordered using *Entry_Time* and *Exit_Time*.

As described in the previous section, two cases were used. In this section, results are provided and discussed.

CASE 1 is associated with one group only. Order By clause was done on the single attribute, then enhanced by the

function call. For the particular sort attribute, no specific index was present. Tab. 2 shows the results taking costs, processing time, and total processed bytes.

TABLE II. CASE 1 RESULTS

One group				
	Order by			
	Single attribute		Function call	
	costs	time	costs	time
	bytes		bytes	
Q11	21	00:00:05	59	00:00:13
	700 B		700 B	
Q12	21	00:00:06	60	00:00:15
	830 B		830 B	
Q13	59	00:00:12	102	00:00:23
	759 KB		759 KB	

Q11 computed the analytical function, which was then filtered. Thus, inner and outer query is necessary. Even though it looks like the most complicated solution from the development point of view, while two queries must be paced, such a solution provides the best results. This is done by encapsulating Order by clause directly in the analytics. Thus, the data set is not physically sorted for consecutive processing. Only the key sort values are treated instead of the whole data source, like specified in the Q12. Although there is no significant difference between Q11 and Q12, a slight difference can be perceived. It is related to the indexing and memory loading. Note that the loading is done on the block granularity instead of the row itself, so the bulk loading is present, reducing the difference in terms of source data sorting and one attribute only. Then, the sorting is operated in the instance memory, which is really fast. The time and cost difference point to that.

CASE 2 extends the previous concepts by splitting the processing into several partitions and restarting the sequence of the analytical function. Another solution is done by the later join. In the inner query, the data amount is filtered using the Fetch first clause. Partitioning is emulated using the Where condition referencing the outer query. This, however, brings a significant performance degradation. Lateral join connects the data dynamically on the fly. It means that the nested query is evaluated multiple times – once for each partition. Thus, instead of taking the whole data set and splitting it into partitions, Lateral join takes only the data part of one partition for processing. It consequently scans the data set multiple times to identify relevant data valid for the particular partition.

Query using analytical function requires 60 costs, while function call takes 63, which refers to a 5% increase. It refers to the function parsing, referencing, evaluating, and context switch between SQL and PL/SQL. Even using UDF pragma does not bring additional performance increase [5]. UDF pragma navigates the system to optimize a function building for the SQL calls instead of a general procedural language environment (PL/SQL).

Comparing analytical functions and Lateral join, it can be clearly concluded that processing partitions separately really does significant performance degradation and rise of the system resources. Costs are significantly increased from the value of 60 for analytical functions to 1239 for the Lateral join. Similar differences were identified for the function reference in the Order by clause. The results for the top data management across partitions are in Tab. 3 and Tab. 4.

TABLE III. CASE 2 RESULTS FOR 1 ATTRIBUTE FOR PARTITIONING

Partitioning – based on a single attribute				
	Order by			
	Single attribute		Function call	
	costs	time	costs	time
	bytes		bytes	
Q21	60	00:00:13	63	00:00:14
	700 B		700 B	
Q22	1239	00:00:22	1273	00:00:22
	830 B		830 B	

Emphasizing the number of attributes used for specifying the partition set, based on the reached results, it can be clearly declared that the partition definition does not impact the performance. There is only a slight difference in the costs, as well as processing time - individual partitions can be processed in parallel, preceded by the splitting operations identifying partitions.

TABLE IV. CASE 2 RESULTS FOR TWO ATTRIBUTES FOR PARTITIONING

Partitioning – based on two attributes				
	Order by			
	Single attribute		Function call	
	costs	time	costs	time
	bytes		bytes	
Q21	62	00:00:13	64	00:00:14
	700 B		700 B	
Q22	1245	00:00:23	1290	00:00:24
	830 B		830 B	

The execution plan for the Lateral join of the partitioned system based on one attribute is depicted in Fig. 5.

```
| Id  | Operation                | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
|   0 | SELECT STATEMENT         |                  |   102 | 35394 |  1239   (4)| 00:00:22 |
|   1 |  NESTED LOOPS            |                  |   102 | 35394 |  1239   (4)| 00:00:22 |
|   2 |   TABLE ACCESS FULL      | FIR              |    34 |  3672 |     3   (0)| 00:00:15 |
|   3 |   VIEW                   | VW_LAT_2D0B8FC8  |     3 |   717 |    36   (3)| 00:00:15 |
|*  4 |    VIEW                  |                  |     3 |   795 |    36   (3)| 00:00:15 |
|*  5 |     WINDOW SORT PUSHED RANK|                |   312 | 33384 |    36   (3)| 00:00:15 |
|*  6 |      TABLE ACCESS FULL   | FLIGHT_DATA      |   312 | 33384 |    35   (0)| 00:00:15 |
```

Fig. 5. Execution plan

For the partition management, there was no specific index definition, therefore the Nested loop operation was selected for the table joining. Index can improve the performance, while the sequential data scanning can be reduced, by shifting the operation from Nested loop to Merge Join. Fig. 6 shows the results for the partitioning by comparing impact of the index definition. For the declarational purposes, costs are emphasized, expressed in percentage.

By analyzing various size of the data sets, the scalability aspect was evaluated. Two additional data sets were used - medium size referred to 500 000 rows in the Flight data table. The large data reference consisted of 5 million rows. The size of the data set does not strictly impact the performance ranges and corresponds with the data number to be processed and evaluated in a linear way. The only difference relates to the Lateral join, which degrades the performance exponentially. The reason is based on the dynamic query evaluation, while the inner query is evaluated for each row of the outer query separately by using the binding condition. The correlation between the costs and size of the data set is stated in Fig. 7. Partitions are defined based on one attribute.

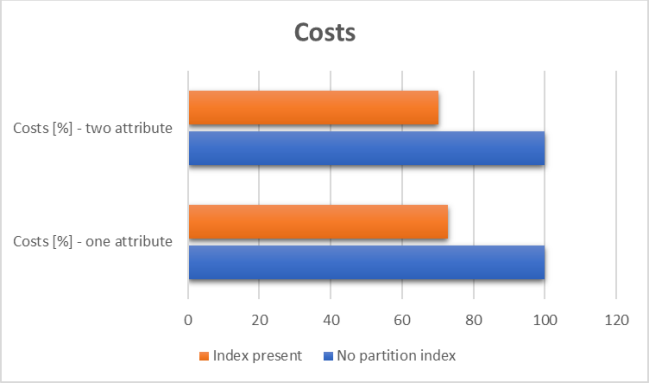


Fig. 6. Execution plan

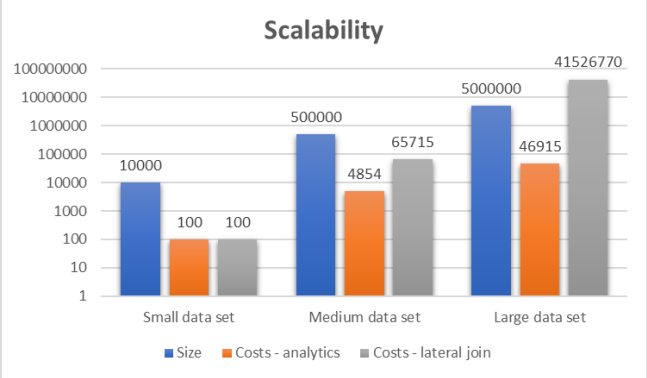


Fig. 7. Scalability

VII. Conclusions

Performance of the data processing, accessing and retrieval is a significant part of the analytics. It is, however, not only important to get the data, it is critically important to get them at the right time and in the right form. This paper is devoted to selecting top data by analyting performance of various solutions, pointing to the whole set or partition the data based on the defined criteria. Despite the fact that today many developers lean towards the Lateral join, as evident, such a soluition does not provide sufficient power for selecting top data., while the groups are dynamically composed and inner query forming the data set for the Lateral join is evaluated dynamically, for each row of the outer query. Analytical functions are much better, Although they process a larger set, in which all relevant data are processed at the beginning, overal performance benefits. This is because the join operations can be better optimized, using the benefits of the indexing, memory pre-loading, etc.

Besides the top data analysis, this paper focuses on the partition definition, as well as the scalability option, whereas the data number to be analyzed is still hugely rising.

In future development, our focus will be on the dynamic index definition, dynamically partitioning data for the analytical functions. We will propose global and local indexes across partitions in the distributed environment. Among that, we are developing own interlayer to let the system postpone joining operation by reducing the data amount to be considered for the analytical function. It is assumed, that the processing can be done in parallel. So, during the analytic function calculation, a dynamic partition index will be created, allowing to join data in partition buckets.

Acknowledgment

It was supported by the Erasmus+ project: 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

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] Hansen, K.: Practical Oracle SQL: Mastering the Full Power of Oracle Database, Apress, 2020

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

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

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

[9] 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

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

[11] Lewis, J.: Cost-Based Oracle Fundamentals, Apress, 2005.

[12] Liu, Z., Zheng Z., Hou, Y. and Ji, B.: Towards Optimal Tradeoff Between Data Freshness and Update Cost in Information-update Systems, 2022 International Conference on Computer Communications and Networks (ICCCN), USA, 2022.

[13] Roske, E., McMullen, T., et. al: Look Smarter Than You Are with Oracle Analytics Cloud Standard Edition, Lulu.com, 2017

[14] Shanbhag, S.: Oracle Cloud Infrastructure 2023 Enterprise Analytics Professional, 2022

[15] 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.

[16] Su S.Y.W., Hyun S.J. and Chen, H.M.: Temporal association algebra: a mathematical foundation for processing object-oriented temporal databases, IEEE Transactions on Knowledge and Data Engineering, vol. 4, issue 3, 1998.

[17] Yao, X., Li, J., Tao, Y. and Ji, S.: Relational Database Query Optimization Strategy Based on Industrial Internet Situation Awareness System, 7th International Conference on Computer and Communication Systems (ICCCS), China, 2022.

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