IJCSI International Journal of Computer Science Issues, Volume 13, Issue 4, July 2016
ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784
www.IJCSI.org http://dx.doi.org/10.20943/01201604.5258

Impact of table partitioning on the query execution performance

Jaumin Ajdari[1], Nehat Mustafa[1], Xhemal Zenuni[1], Bujar Raufi[1], Florije Ismaili[1]

[1] Faculty of Contemporary Sciences and Technologies
South East European University (SEEU)
Tetovo, Republic of Macedonia

Abstract

With the increase of database, it is commonly to have tables with a considerable number of data records. This will bring delays in database use specially in query execution. A good way to improve performance and reduce the time is by applying the partitioning technique, which are known as techniques that restructures the table data records. Partitioning is a good way to simplify database maintenance, management and administration and as well as working with queries. A very important advantage of using partitioning is that it allows to query and scan just a small part or some parts of partitioned tables. In this research we examine the use of partitioning techniques on relatively large table and we measure the query execution time and cost. As a result, we have shown some comparisons and given suggestions how to further increase the performance.

Keywords: *Partitioning, Query execution time, Partition pruning, Paper Specifications, Partition-wise join.*

1. Introduction

While increasing the data volume in the database, it is common to have tables with considerable number of data rows. This will bring delays in implementing the requirements and use of data, as a result, it will decrease the performance. A good way of performance improvement and reducing the time of data use is the implementation of partitioning, which is known as a technique that restructures the data in the table. Partitioning is presented as a technique for managing large objects and in our case tables. Maintenance, management and administration of the database with large tables is a new challenge where one of the methods for resolving this challenge is through the use of partitioning. Partitioning resolves this by dividing the table into smaller parts, which allows having an easier management and administration. Another important advantage of the partition use is that queries have quick access to the data. For example, when doing a search query in the table, it won't do a scan of the whole table, but it will scan just some parts of the table according the partitions.

The main purposes of our paper are:

1. Analyzing how partitioning techniques can help on managing large tables and impact of partitioning in data access and update
2. Analysis and comparison of the use of large tables, when table is with and without partitions
3. Building a guide for using partitions
4. Presentation of the results of the analysis and the implementation of the method of partitions on different cases, which would indicate new opportunities in the use of database.

In this paper we will examine the performance improvement with use of the partition techniques of the large tables. We will explain the partition techniques used on an Oracle managed database and at the end we will present the results obtained with the query execution testing.

2. Related work

In recent years, many papers have been published where the benefits of the database performance by using the method of partitioning are analyzed and explained. In the follow we mention some of those results. The authors Herodotos Herodotou, et al., have explained that the trend of rapid growth of data has forced the use of partitions in databases. According to their paper [1] the creation and correct use of partitioning enables a better database performance by reducing the number of unnecessary data from the query processing, parallel data access during query execution and easier data management and maintenance.

With continuous database increase, not only tables are increased but at the same time we have increase of index tables. Same concept of table partition can be applied for indexes. According to Eugene Wu and Samuel Madden [2] although indexes are standard method for query performance improving, but sometimes it isn't enough just use of indexes. According to them [2] increase of data volume will increase the index tables and decrease the performance of data insertion. The authors suggest index

partitioning. In research [2], it is analyzed the use of partitions and optimizing the indexes by dropping the indexes and selectively index just the partitions that are accessed by queries. The performance improvement is evidently compared to the traditional use of the indexes.

Mayur Sawant, et al. [3] and Abhay Kumar and Jitendra Singh Yadav 11] in details have described the techniques of table partitioning. Paper [3] is focused on the three key methods of partitioning and the composite partition strategies which includes the date, range and hash partitions. In both papers is explained that the partitioning strategy helps to reduce the delay in response time.

Zoltan Mathe and Philippe Charpentier [4], have shown their experience in coping with huge amount of data by use of partitioning techniques. They have used the composite partitioning strategy such as range-hash partition, partition pruning and usage of the Partition-Wise joins. The achieved result is that with use of partition pruning in case of store a large amount of data, it essentially increases the database performance as a result of reducing the amount of data retrieved from the disk and optimizing the resource utilization.

Researches [7, 8, 9, 10] have analyzed partitioning technics focused on distributed databases and query executions as well as the strategies and technics and also advantages and disadvantages of use of partitioning.

Optimization of query plays an important role in the performance of the database, especially when executing a complicated SQL command. Another issue is to improve the performance query when access a large amount of data. According to many researches [1, 2, 4], partitioning limits the amount of data to be examined and analyzed, enabling analyze of a part assigned to the table instead of the table as a whole, so when the query retrieves data from the table, it doesn't do a full table scan as a whole, but only a partition or several partitions.

3. Partitions

With the growth of the database it is common to have tables that have a significant number of data rows. This will bring delays in the implementation of request in the database. A good way to increase performance and reduce the time of these request is the use of partitions, is known as a technique that restructures the data in the table. Partitioning [5, 12] of a table means division of a table in more parts and maintaining the view as if it were a single table. The parts are called partitions and each of partitions should share the same columns, constraints, indexes and triggers. Also partitions can have its own unique parameters of storage, so developers can choose to deal with the entire table or individual partitions. Partitioning allows us to divide the rows of a table based on a logical expression of a column called as partitioning key. It consists of one or more columns that sets partition.

The partition is one of the most efficient methods to ease the problems database maintaining, limits the amount of data to be reviewed and analyzed just by use a certain part of the table instead of the table as a whole. Partition allows the administrator to apply the method known as "divide and conquer", data management improve. The partitions also enable to perform some specific activity, rather than those activities performs in general, such as making a backup to a partition instead of whole table, or making any change only to a partition instead into whole table. It is easier and faster to delete records because we work with a part of the table instead of whole table, and also it is much easier to delete a partitions by using DROP PARTITION command instead of using the DELETE command into whole table. Another advantages of partitioning is when we work with queries. Partitioning enables the query to scan any parts of table and not the whole table it means that will be scanned one or few partitions. So the search is made in smaller parts or in separate parts of the table. In the following we will focus in Oracle partitioning techniques, Oracle contains two partitioning methods, which are known as partition pruning and partition-wise join.

Partition pruning is a relatively simple concept of partitioning, which can be described as "do not scan partition where there can be no matching values." Partition pruning enables to access only that part of the table that contains the necessary data, that the query have requested, enabling the choice only of the part that we want [4, 5]. In other words, partition pruning is sometimes referred to as partition elimination, because eliminates, or ignores partitions that are irrelevant to the criteria that we have set in SQL query.

Partition-wise join is essentially partitioning which can improve the performance of multi-table joins, by using a technique known as partition-wise joins. Partition-wise join is an optimized join of two or more partitioned tables, which can be implemented and take effect of optimization when the join tables or indexes using partitioning key as join attribute between tables [4, 5, 13]. Partition-wise join breaks large and complex join operations into smaller join operations, which can then be used sequentially or parallel.

Another advantage of using the partition tables is that each partition or object is independent to each other. For example, if one partition of a partitioned table is

unavailable, the all other partitions of that table remain available. The application can continue to use query and perform transactions to the available partitions of the table, and DBMS will successfully execute operations while those operations will not need to access the unavailable partitions. The other issue is that with increasing database, it does not impact just in increasing of the data table but this also increase the index tables. Same concepts of the table partitioning can be applied into index tables. Index partitioning allows greater flexibility in terms of how users and application will access the data. The great benefit is that the query engine will analyze only the index partitions which are needed to respond to the query, so in considerable way will be increase the query access speed. Partitioned index is a simply index divided into smaller parts that can be stored on different disks (reducing input/output operation). Both method of indexes B-tree and bitmap can be partitioned. All available partitioning strategies rely on:

1. **Fundamental data distribution methods** that can be used for either single (one-level) or composite (two-level) partitioned tables.
2. **Partitioning extensions**

Oracle partitioning offers three fundamental data distribution methods that control how the data is placed into partitions. The data distribution methods are range, list and hash partitioning. The first partition method, used by Oracle in the Oracle version 8 is the range partition. Range partition method splits the table on parts which contain rows according to range of values of one or more table columns [14]. The simplest and widely implemented of range partitioning is the case where column or attribute domain is used as a partition key range and this is the commonly used method in Oracle. Range partitioning method is more appropriate for use when the table contains historical data.

List partitioning is similar to range partitioning in many ways. The main difference between the two types of partitioning is that, the range partitioning is defined on the basis of the neighbors range, and in list partitioning, each partition is defined and selected based on the membership of a column value in one of a set of value lists. List partitioning often comes in use when the table which needs to be partitioned contains any code or ID as a column value (id of states, cities, departments, regions, etc.) [17].

Partitioning by hash is commonly used to ensure data delivery between a predetermined numbers of partitions. Hash partitioning with use of well-known hash algorithm will make approximately equal distribution of data between partitions, creating partitions approximately the same size. The recommendation is that the number of partitions, on general, has to be a power of number two (2, 4, 8 16 and so on), so, the hash partitioning algorithm to achieve a better data distribution [15]. On hash partitioning we have no control over the hash algorithm or how Oracle distributed data, the only job we have to do is to determine the number of partitions in which we want to have the table.

Composite partitioning is a composition of the above mentioned partitioning methods and it can be partitioning list – range, range – range, range – list, list – hash, list – list methods. Composite partitioning means combination of the two partitioning methods. In the latest versions of Oracle DBMS, Oracle offers a new strategy, known as Partitioning Extensions, which makes the extensions of the basic partitioning strategies by offering more flexibility in defining of table partitioning key. Partitioning extensions contains the partitioning methods known as interval partitioning, reference partitioning and virtual column-based partitioning.

4. Analysis and results

To analyze the use of partitions and to see the advantages of partitions, we made some experiments. As experiments we execute some queries on a database which contains the tables with several million records (about 50 million records and a size of 2.6 GB, database with information from the Pension and Disability Insurance Fund of Republic of Macedonia). Our analysis and testing in this paper are focused on the query execution time and cost in three different cases:

1. comparing the use of the table with and without partitions,
2. comparing the use of the multi – join with and without partitions tables and
3. comparing the use of the table with partitions and without partitions but indexed.

4.1 Analysis of query execution on table with and without partitions

In case of the analyze the differences of a query execution time and cost, we execute a query which use only one table, the table PAYMENTS, when the table is not partitioned and partitioned. The query gives data about monthly pension payments for a certain period and other certain conditions defined in the WHERE clause. This query will access only the PAYMENT table. Query will be executed in three cases:

1. in the table with 1 million rows

2. in the table with 3 million rows
3. in the table with 20 million rows

In the following tables are shown the summarized results after execution of query.

Table 1. Query execution time

No. of records	Without partitions	With partitions
1 million	2.641	1.687
3 million	8.391	5.859
20 million	35.672	25.016

Table 2. Query execution cost

No. of records	Without partitions	With partitions
1 million	5255	3566
3 million	8869	5711
20 million	88679	35664

The following figures clearly shows the difference in the implementation of query when it is used partitioned and without partitioned table.

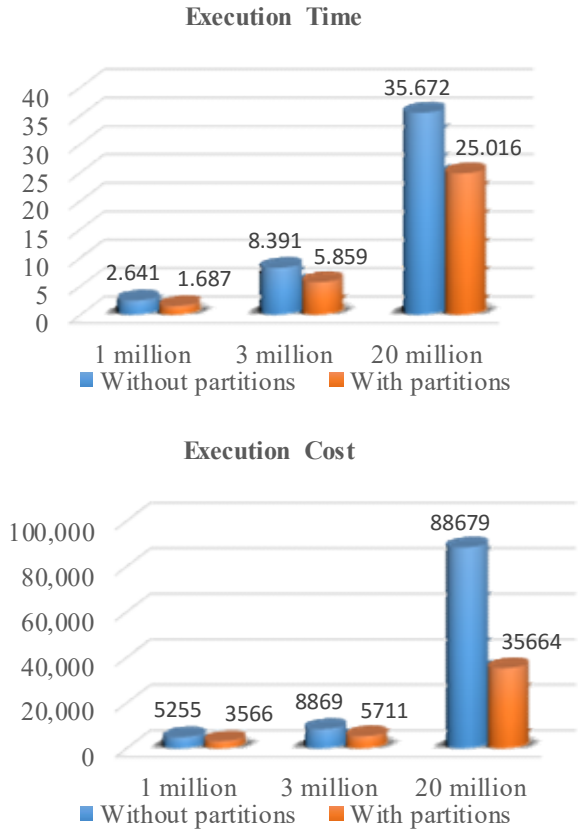

Fig 1. The effects of using partitions

According to the obtained results we see a significant performance improvement in case of partitioning use. Indeed, in case of the table with one millions of records the improvement is around 56% to the execution time (from 2.641sec to 1.687sec, the difference 0.954 sec) and 47% to the cost (5255 – 3566 = 1689). In case of the table with three millions of records the improvement is around 43% to the execution time (from 8.391sec to 5.859sec, the difference 2.532sec) and 55% to the cost (8869 – 5711 = 3158) and in case of table with 20 millions of records the improvement is around 43% to the execution time (from 35.672sec to 25.016sec, the difference 10.656sec) and more than 100% to the cost (88679 – 35664 = 53015). Therefore, we can conclude that with the use of partitioning we have significant performance improvement of query especially when the number of records in the table is very large.

4.2 Analysis of query execution when we have multi – join tables

Our second query provides data from two tables, from the monthly payments table and from one additional payments table which contains additional monthly payments to pensioners which are paid out of regular monthly pension. So, the query has access into two tables (monthly payments table with 20 million of records and the extra payments table with 3 million of records) and it uses the join between those tables. In the following table are shown the summarized results after query execution.

Table 3 The effects of using wise-join partitioning method

	Without partitions	With partitions
Time	**15.859**	**0.828**
Cost	**57807**	**756**

The figure 2 clearly shows the difference in the implementation of query when we have joined table with and without partitions.

Analyzing the obtained results, we see enormous performance improvement on both measures, execution time and cost. The execution time has a better improvement over 19 times (from 15.859sec to 0.828sec and difference 15.031sec) and the execution cost over 76 times (57807 – 756 = 57051). According to the obtained results, we can conclude, that using the partitioning improves join queries even if the queries are compounded from many tables in join.

From the results above, it is clear that by use of the partitioning, we achieve significantly improved performance. Improvement is seen in both cases, in case of query execution on a single table and more improvements in case of query execution on two or more linked tables. This improvement comes as a result of the data accessing, so the use of partitioning enables access to selected parts

and does not require access to the entire table. In the next, we analyze the effect of partitioning versus indexing.

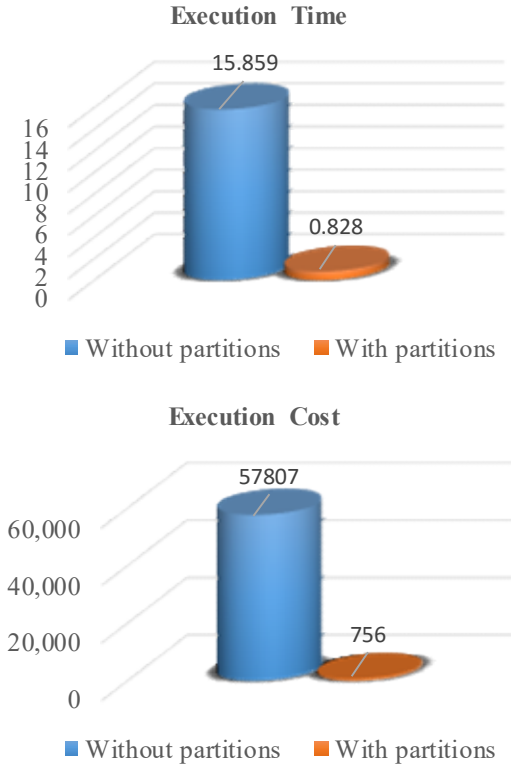

Fig. 2. The effects of using wise-join partitioning

4.3. Analysis of query execution on partitioned and indexed table

As it is known, the index enables quick access to the data in the table and indexes created for the attributes that appears on the selection condition speeds up the query execution. Below we will analyze the speed up as a result of indexing and comparing to the speed up gained by use of partitioning. To show a more visible comparison, we execute a query in a table with around 50 million records (and size about 2.6GB), the table where are stored the data about contribution payments of the pension insurance. The query is executed for the time period of 2008 year and it is executed in two cases, first time for the period from January to March and the second time for period between January and December. As a result, the query returns the total number of insured persons to payment their contributions for a given period and the result for the first time was 657060 selected records and 13010545 selected record for the second time. The same query is executed in both cases, when the table is partitioned (and the partition key attribute is used in the selection condition) and the second case when the table is indexed by the attribute which is used in the selection condition. In the following tables are shown the summarized results of query execution.

Table 4. The obtained query execution time while using tables with indexes and tables with partitions

Case	Table with indexes	Table with partitions
1	0.109	4.079
2	13.297	4.047

Table 5. The obtained query execution cost while using tables with indexes and tables with partitions

Case	Table with indexes	Table with partitions
1	1200	15352
2	36610	15374

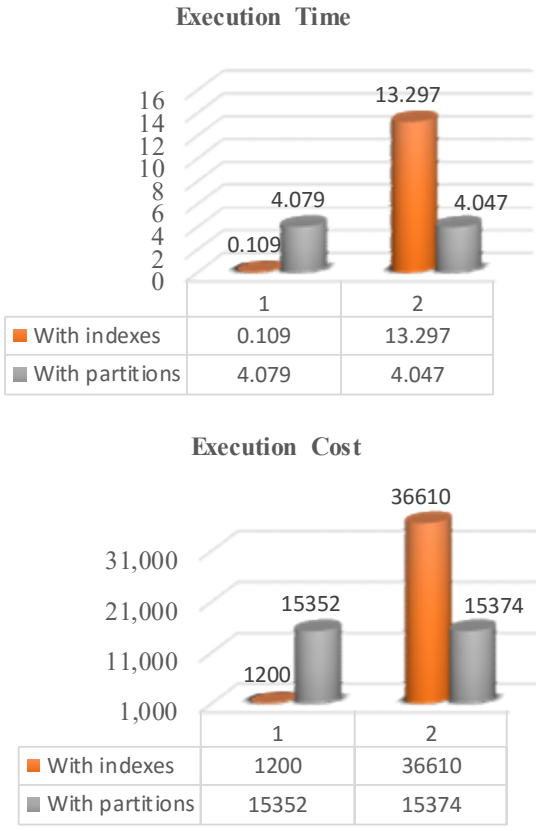

Fig. 3 Results obtained of query execution time and cost while using indexed table and partitioned table

Analyzing the obtained query execution results (comparing number of returned records with total number of records and execution time and cost), from both cases, we see the significant difference between times and costs. In first case (case 1) where the query as a result returns 657060 records, the best results we have with use of indexes. While in second case (case 2) where the query returns 13010545 records, we see the opposite, so in this case the execution time and cost are lower to partitioned table than

indexed table. According to the above results, we can confirm and conclude that indexed table is more appropriate when the query result is up to 5% of the total number of records (as in case 1) and in the case where the query result is greater than 5% then it is preferable partitioning use (as in case 2 where the result is around 26%). So, in general, if it expected access to small amount of data records (comparing to the whole table) then indexes are the best choice, and the partitioning gives more benefits when as a query result is expected a relatively large amount of data records.

5. Conclusions

Nowadays databases become larger and larger. One of the main problem is maintaining those databases and coping with large tables. Another problem of database increase is the fact that proportionally to database increase the query execution performance decrease due to query access on more data, and it brings respond time delays. One of the way how to deal and maintain large tables is to use partitioning technics. Easy implementation, simple administration, less complex are some of the advantages that make partitioning one of the best strategy and solution and the one of frequently used in the database, especially in the "very large database". Regarding the query execution, partitioning in some way optimize the query because query access just a part of table and not the whole. This optimization of the query significantly will improve the query and database performances.

In our case, we want to show the database performances improvement as a result of partitioning. In an Oracle database with relatively large tables (in experiments we have used tables with a few million records, up to 50 million records) we execute some queries and we measure the execution time and cost. We compare the query execution on case with and without partitions and also comparison between using partitions and traditional indexes. Partitioning simplifies these problems by limiting the amount of data to be reviewed, analyzed, enabling review of a specific part of the table instead of as a whole table. This mode also enables the query to scan just one or some partitions instead of table whole scan.

Through our experiments and the obtained results, we prove that the use of partitioning when we use table with large number records shows the increase of query execution performance. We achieved this by partitioned table according to the attribute which is used in condition of selection. From the results it is seen that the performance improve exceeds 50%. Also the comparison between the use of indexes and partitioning is significantly. So, if expected returns, the number of return records is up to 5% then it is preferable to use indexes and in the opposite use of partitioning technique.

Acknowledgments

We want to thank the Pension and Disability Insurance Fund of the Republic of Macedonia which gave us the possibility of using their data and database and to conducting experiments.

References

[1] Herodotos Herodotou, Nedyalko Borisov, Shivnath Babu, Query Optimization Techniques for Partitioned Tables. Conference: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, pp. 49-60, Athens, Greece, June 12-16, 2011

[2] Eugen Wu, Samuel Madden, Partitioning Techniques for Fine-grained Indexing, Proceeding ICDE '11 Proceedings of the 2011 IEEE 27th International Conference on Data Engineering, Hannover, April 11-16, 2011, pp. 1127-1138

[3] Mayur Sawant, Kishor Kinage, Pooja Pilankar, Nikhil Chaudhari, Database Partitioning: A Review Paper, International Journal of Innovative Technology and Exploring Engineering (IJITEE), ISSN: 2278-3075, Volume-3, Issue-5, October 2013, pp. 82-85

[4] Zoltan Mathe, Philippe Charpentier, Optimising query execution time in LHCb Bookkeeping System using partition pruning and Partition-Wise joins, 20th International Conference on Computing in High Energy and Nuclear Physics (CHEP2013) IOP Publishing Journal of Physics: Conference Series, Volume 513 (2014), Track 4 (042032).

[5] Oracle White Paper, Parallel Execution with Oracle Database 12c Fundamentals. Oracle White Paper 2014, pp. 1-5.

[6] Mayur Sawant, Kishor Kinage, Pooja Pilankar, Nikhil Chaudhari, Database Partitioning: A Review Paper, International Journal of Innovative Technology and Exploring Engineering (IJITEE), ISSN: 2278-3075, Volume-3, Issue-5, October 2013, pp. 82-85

[7] Sudesh Rani, A Partitioning strategy for OODB, IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 6, No 3, November 2011 ISSN (Online): 1694-0814, pp. 317-321

[8] Alka Gangrade and Ravindra Patel, Performance Analysis of Privacy Preserving Naïve Bayes Classifiers for Distributed Databases, IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 2, No 3, March 2013 ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784, pp. 423-429

[9] Seema Patil, Prof. H. M. Jadhav, Distributed query execution system for Transactional Database using Lookup Table, International Journal of Emerging Technology and Advanced Engineering, ISSN 2250-2459, Volume 3, Issue 2, February 2013, pp. 125-131

[10] Herodotos Herodotou, Nedyalko Borisov, Shivnath Babu, Join Optimization Techniques for Partitioned Tables, Proceeding SIGMOD '11 Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, Pages 49-60

[11] Abhay Kumar, Jitendra Singh Yadav, A Review on Partitioning Techniques in Database, IJCSMC, Vol. 3, Issue. 5, May 2014, pp.342 – 347
[12] Sanjay Mishra, Alan Beaulieu. Mastering Oracle SQL. 2nd. ed. O'Reilly Media, June 2004, ISBN: 978-0-596-00632-7
[13] Murali Vallath, Oracle Real Application Clusters. Digital Press, 2003, ISBN :9781555582883
[14] Darl Kuhn, Pro Oracle Database 12c Administration, 2nd Edition , Apress, 2013, ISBN13: 978-1-4302-5728-8
[15] Tomas Kyte, Expert Oracle, Signature Edition Programming Techniques and Solutions. Apress, 2005, ISBN 10: 1590595254 / ISBN 13: 9781590595251

Jaumin Ajdari, Assist. Prof. at Faculty of Contemporary Sciences and Technologies, South East European University. His current research interest is in parallel processing, data processing and databases.

Nehat Mustafa, MSc., Faculty of Contemporary Sciences and Technologies, South East European University. His current research interest is databases and data processing.

Xhemal Zenuniu, Assist. Prof. at Faculty of Contemporary Sciences and Technologies, South East European University. His current research interest is in semantic web, contemporary distributed systems, intelligent agent and data analytics.

Bujar Raufi, Assist. Prof. at Faculty of Contemporary Sciences and Technologies, South East European University. His current research interest is in adaptive web, semantic web, computer graphics and data analytics.

Florije Ismaili Assist. Prof. at Faculty of Contemporary Sciences and Technologies, South East European University. His current research interest is in web services, cloud computing and information retrieval.