DOI: 10.15439/2022F154
ISSN 2300-5963 ACSIS, Vol. 30

Flexible user query order for the speculative query support in RDBMS.

Anna Sasak-Okoń
Maria Sklodowska-Curie University in Lublin
Pl. M. Curie-Skłodowskiej 5, 20-031 Lublin, Poland
Email: anna.sasak@umcs.pl

***Abstract*—This paper concerns speculative query execution support for RDBMS based on the dynamic analysis of input (user) query stream. A middleware called the Speculative Layer is presented. Based on a specific multigraph representation of groups of consecutive input queries, called the Speculation Window, the Speculative Layer generates speculative queries for look-ahead execution. These speculatively obtained results are then used while executing user queries. This paper shortly presents the structure of the Speculative Layer and the adopted graph modelling method. Then, a new strategy of queries in the Speculation Window is introduced. Depending on the availability of executed speculative queries results we allow order of user queries in the Speculation Window changes. If a user query was to be executed without the speculative support, we prefer to delay its execution in favour of one of the consecutive user queries, expecting that speculative results obtained in the nearest future will by useful for the delayed query. The experimental results presented in a multithreaded environment, cooperating with a SQLite database, show that the proposed strategy reduces the number of user queries executed without the speculative results. Additional series of experiments verifies that the certain parameters describing the speculative support system, like Speculation Window size, are properly chosen.**

I. Introduction

SPECULATIVE Parallelization, sometimes called Optimistic Parallelization[1] is a technique which allows parallel execution of code fragments that were originally intended for a sequential run. To ensure correctness, speculative results must be verified to avoid dependence violations. In such case a violating thread and its results are usually discarded and restarted with updated data [7][8][9]. The practical use of the speculative approach in relational databases usually supports single queries, ranked queries [14] or transactions (speculative transaction protocols) [12][13] by performing some anticipated, potentially useful operations or subqueries out of its standard order [10] or in advance based on received earlier tips [11]. There are also papers, like [15][16], which focus on multiple query optimization although without the concept of the speculative execution.

Graph structures often used as a formalism helping represent and analyse queries are very popular as they naturally represent and model entities and their relationships [17][18].

It should be noted that none of the described above optimization methods aims at speculative support which covers need of many future queries at a time. Saving the results obtained speculatively reminds caching methods [19][20][21][22], but instead of history based methods we prefer to analyse queries which are waiting for execution in the nearest future and support them with data prepared in advance. The speculative execution model we introduced in our previous papers [2][3][5][6] focuses on parallelised speculative support for execution of input (user) query streams. It includes a multithreaded middleware called the Speculative Layer, which is situated between user applications and the RDBMS. The aim of the Speculative Layer is to choose and execute speculative queries. These speculative results stored in the main memory structures called Speculative DB constitute quick access data set available while executing user queries. The process of choosing speculative queries to be executed is called the Speculative Analysis. The Speculative Analysis is performed continuously for groups of consecutive user queries called the Speculation Window (SW). For each Speculation Window a specific graph representations of each user query are created, which are then combined into one multigraph according to the defined set of rules.

The Speculation Window (SW) moves over the user query queue by one query. So far, for each SW, the first in order user query was executed nonspeculatively with or without the use of speculative results. In this paper we propose a new strategy for a nonspeculative query execution which is dependent on the availability of the speculative results. If a first query in the SW would have to be executed without speculative results we allow to replace it with a next query in the SW which has at least one executed speculative query assigned. We assume that one of the speculative results obtained in the nearest future will be useful for the delayed query. Speculation Window moves after the execution of the nonseculative query and we repeat the attempt to execute the first user query nonspeculatively.

The remaining text of the paper is composed of 3 parts. In the first part a general structure of the Speculative Layer is presented. We describe rules for query graphs creation and the process of Speculative Analysis resulting in optimally defined set of speculative queries ready for use. Section III describes the new strategy for the nonspeculative query execution. Section IV contains results of two series of experiments. First series of experiments presents effectiveness comparison between the Speculative Layer execution with old and new strategies for the nonspeculative query execution. The second series of experiments verifies the validity of the parameter describing the size of the Speculation Window chosen for the previous versions of the speculative algorithm.

II. The Speculative Layer

Fast evolution of online activities induced development of database applications with the specific characteristics. These applications are intensively used as products browsing tools and thus execute many queries of a specific structure. Such queries are created by shop users who define search criteria, directly influencing attributes and conditions in the SELECT and WHERE query clauses. Therefore, the consecutive queries contain some common constituent operations, whose results, if executed speculatively can be used many times.

The above observations were an inspiration to propose a dynamic speculative support for execution of sql user queries. This model is implemented as an additional multithreaded middleware called the Speculative Layer and is located between user database applications and the RDBMS. The stream of user queries forms a queue which is continuously analysed by the Speculative Layer. The analysis (Speculative Analysis) is performed for the consecutive user queries grouped in a structure called the Speculation Window (SW) which slides on the query stream. Each user query is represented by its query graph created with a set of defined rules. Then, the single query representations for each Speculation Window are merged together to create a joint representation of the whole query group in the form of query Multigraph (Q_M).

The Speculative Layer is implemented to support and analyse CQAC (Conjunctive Queries with Arithmetic Comparisons) queries with functionality extended by two more allowed operators: IN for value sets and LIKE for strings comparisons. Additionally we allow a nested query in a WHERE clause returning a value for the attribute condition (...WHERE...attribute operator (SELECT...FROM...WHERE...)).

The aim of the Speculative Analysis process is to identify some common constituent operations in user queries which are then used to generate new queries, called the Speculative Queries. For this, an extended version of Q_M is created called Speculative Query Multigraph (SQ_M). It contains speculative edges which are a special type of edges which mark potential speculative queries. Based on the speculation result we introduce two types of speculative queries: **Speculative Parameter** or **Speculative Data** Queries. As there is allowed possibility of the modifying query occurrence in the query stream, and thus in the Speculation Window, we introduce additional type of speculative edges called **Speculative State**. Details of the strategy for modifying query handling process can be found in [6]. Fig.1 presents the SQ_M created for the following component queries:

(Q_1) SELECT $A_{1,2}, A_{1,3}$ FROM R_1 WHERE $A_{1,2} = 4$
(Q_2) SELECT $A_{2,2}, A_{2,3}, A_{1,3}$ FROM R_1, R_2 WHERE $A_{1,4} = A_{2,1}$ AND $A_{2,2} > 7$
(Q_3) SELECT $A_{1,2}, A_{2,2}$ FROM R_1, R_2 WHERE $A_{1,4} = A_{1,2}$ AND $A_{1,3}$ LIKE $'\%xx'$ AND $A_{2,2} < 2$
(Q_4) SELECT $A_{2,2}, A_{2,3}$ FROM R_2 WHERE $A_{2,2} > 5$

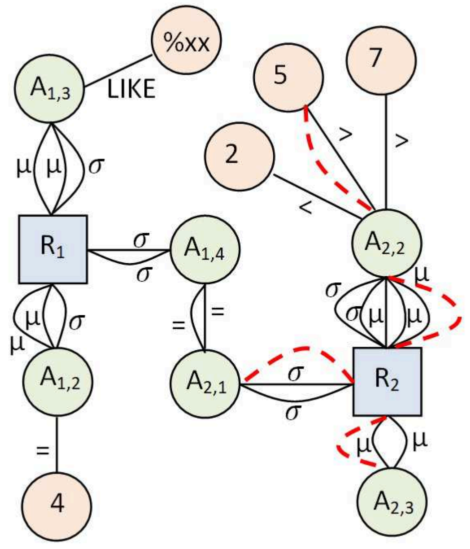

Fig. 1. Query Multigraph representing four user queries.

It has also Speculative Data edges inserted (red stripped lines) which represent one of 5 possible speculative queries to be executed for this SW: (sq_1) SELECT $A_{2,1}, A_{2,2}, A_{2,3}$ FROM R_2 WHERE $A_{2,2} > 5$. Such speculative query results, if executed, can be used while executing two user queries from this SW: Q_2 and Q_4.

Chosen speculative queries are executed in parallel with the original user queries and its results are stored in a dedicated RAM memory database structure called the Speculative DB. The data from the Speculative DB is ready to be used during execution of input user queries, called nonspeculative queries, minimizing disk database reads.

The original user query executed for each Speculation Window (so far it was always the first query in the SW) is called the nonspeculative query. If there are executed speculative queries assigned to the nonspeculative query it is transformed to use them, otherwise it is executed without the speculative support. Each user query can be executed with the use of the speculative results prepared for each relation present in its FROM clause. More details about algorithms implemented for query graph manipulation and strategies of multiple speculative queries results combining for the execution of one user query can be found in [4] and [3]. After the nonspeculative query's results are returned to the user, the SW moves by one query. As a result, the representation of the executed user query in the Q_M is replaced by the representation of the next user query from the queue, which just entered the Speculation Window. The SW move is followed by the aforementioned Speculative Analysis process to generate a new group of speculative queries for execution. Described operational scheme is repeated until there are user queries waiting for execution.

Based on previous experiments presented in [2] the Speculation Window size was set to 5 and the number of active threads for each SW was set to 3 (bigger SW or more threads didn't provide further improvement in user queries execution). One thread is always dedicated to the execution of the nonspeculative user query, so the remaining two threads can execute chosen Speculative Queries. From the group of awaiting Specualtive Queries generated in the process of the Q_M Speculative Analysis, two of them can be chosen for execution. The highest execution priority is always assigned to speculative queries which can be used by the highest number of user queries. Additionally, we consider values of two defined size reduction metrics for speculative queries - Vertical and Horizontal Selectivity (for definition see [5]). As we want to avoid creating full copies of database relations in the RAM memory we look for speculative queries with possibly low values of these metrics. If it was not possible for a particular Speculation Window to choose a new speculative query for execution, the speculative thread would report a nojob status for this SW. Executed Speculative Queries are registered on the list and assigned to user queries which can then use their results.

III. A NEW STRATEGY FOR THE SW EXECUTION

The old execution strategy for the Speculation Window (SW) was fixed, id est. the nonspeculative query was always the first query in the SW. This way, we kept the execution order of user queries risking that for some of them the speculative results might not be ready yet.

New strategy allows the nonspeculative query not to be the first one in the SW. If the first query in the SW would be executed without the use of the speculative results, then a next query in the SW which can be executed with already obtained speculative query results is executed nonspeculatively. This way, the speculative algorithm has a chance to prepare and execute new speculative query/queries whose results will be beneficial for the delayed user query. The expected execution time reduction provided by the use of the speculative results should outweigh the potential execution delay.

Such situation is presented in Fig.2. We can see a Speculation Window containing 5 user queries (Q_2-Q_6). Below each blue user query square, there are orange circles containing ids of the executed speculative queries assigned to a particular user query for the potential use. The first query in the SW has no executed speculative queries assigned (no circles below). Thus, we look for the next query in the SW which could be executed with the use of the speculative results. The Q_3 user query is then executed nonspeculatively with the use of one of two assigned speculative queries (in certain cases it is possible to use more than one speculative result for one user query). Then SW moves, the nonexecuted Q_2 remains in it, while the Q_3 is replaced by the next query from the user query queue.

IV. EXPERIMENTAL RESULTS

A. Test Environment and Queries

The Speculative Layer is implemented in C++ with the multithreaded execution with the Pthread library. The SQLite 3.8.11.1 engine is used as a database management system. For the experiments we used the database structure and data (8 relations, 1GB data) from the well known database benchmark TPC-H [23]. Additionally, a set of 8 Query Templates was prepared and used to generate 3 sets of 1000 input queries each, with random values used in the attribute conditions in WHERE clauses. Structures of the T1-T8 templates are listed below with the following notation: TemplateId:RelationName(number of WHERE conditions referring to its attributes,...,RelationName(number of WHERE conditions...)

T1: LINEITEM(1 with a nested query)
T2: LINEITEM(4), PART(2)
T3: PART(3), PARTSUPP(1)
T4: LINEITEM(4), ORDERS(3), CUSTOMER(1)
T5: LINEITEM(3), PART(4), PARTSUPP(1)
T6: LINEITEM(3), ORDERS(1), CUSTOMER(1), PART(2)
T7: LINEITEM(3), ORDERS(1), CUSTOMER(1), PART(2), PARTSUPP(1)
T8: UPDATE ORDERS

Templates T2-T7 are Select query templates which join from two to five database relations. Template T1 refers to one relation but includes a nested query in its WHERE clause. T8 represents modifying queries. Each test queries set contains approximately 4% of modifying queries and 96% of select queries with the same density for each of T1-T7 templates.

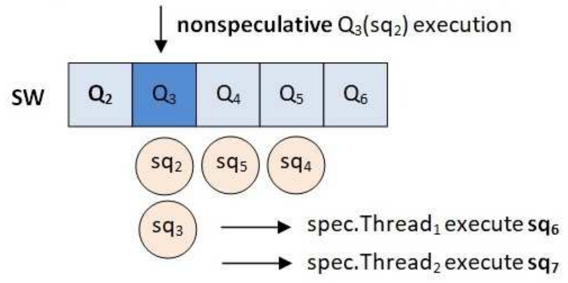
nonspeculative $Q_3(sq_2)$ execution
SW
Q_2
Q_3
Q_4
Q_5
Q_6
sq_2
sq_5
sq_4
sq_3
spec.Thread$_1$ execute sq_6
spec.Thread$_2$ execute sq_7

Fig. 2. New nonspeculative query execution strategy.

B. A New Nonspeculative Query Execution Strategy

First, we have compared how the new nonspeculative query execution strategy for the Speculation Window affects the execution of user queries. For this, we compare the results obtained with the old execution strategy (nonspeculative query is always the first one in the SW) with the new strategy (nonspeculative query is the first query in the SW which can use already obtained speculative query results). The experiment was conducted for the Speculation Window size=5 and 3 active threads. Fig.3 presents average execution times obtained for each query template depending on how many speculative query results were used (from 0-red bars to 3-yellow bars speculative queries results for one user query). We can see that each speculative query used, provides further reduction in the user query execution time. The highest speedups are obtained for T5 and T7 templates. These user queries have considerably longer execution times if executed without the speculative support (approximately 163 and 181 sec.). With

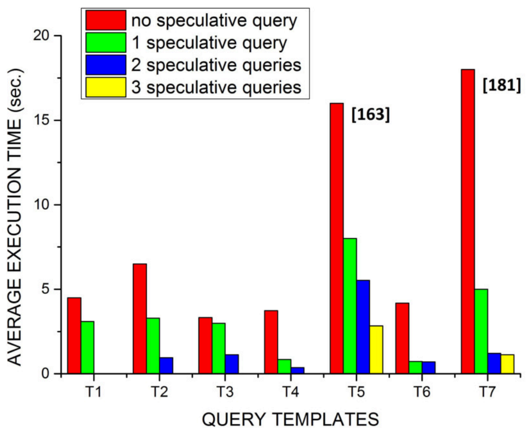

Fig. 3. The average execution time for 7 query templates with and without use of the speculative query results.

three speculative queries results used (one speculative query for each relation appearing in a query) we managed to significantly reduce their execution times (up to 9 times shorter). For the remaining templates the average time reduction is approximately 2 times. We have next studied how the new strategy influenced the general numbers of queries executed with or without the speculative support. The new strategy reduced the number of user queries executed without the use of the speculative result from approximately 113 to 57 (for 1000 user queries), which is almost two times less. The query order change (id est. as a nonspeculative query was executed a user query not being the first in the SW), was reported 132 times and concerned 58 user queries. The maximal delay of the first query in the Speculation Window execution was 4, with average delay for the whole set equal 2.59.

C. Speculation Window Size

As the implementation of the new strategy for the nonspeculative query choice provided considerable reduction in the number of user queries executed without the speculative support, we decided to run a new series of experiments to verify, if the Speculation Window (SW) size change can provide any further benefits. Table 1 presents the results we obtained for the SW sizes from 5 to 15. The size of the SW equals the number of user queries used to form a Query Multigraph. We can see that with the growing size of the SW, the number of nojob reports is also growing significantly, up to 232 for the SW size=15. The reason for this is that with bigger Speculation Windows (and bigger multigraphs) sizes, each SW move is still replacing only one user query in the multigraph structure. Such structure change is not enough to generate new and unique speculative queries for execution. Thus we can see, that the total number of executed speculative queries for the test set is actually decreasing. Even though each executed speculative query is assigned to more user queries for potential use (on average 1.98 for SW size 5 to 3.56 for SW size 15) it doesn't influence (preferably reduce) the number of user queries executed without the speculative support. The reduced number of queries obtained for SW=5 decreases further for SW size 6 to reach an average value around 33 for bigger Speculation Windows.

Four bottom rows of Table 1 present results characteristic for the new strategy of nonspeculative query execution. We can see that the number of the user queries whose execution were delayed due to missing speculative query results, varies around 60 and doesn't seem to be dependent on the SW size. However, the bigger SW gets, the execution delay of such queries grows up to 13 steps for the SW size 15, which is not an advantageous feature. Too long delays would require a new mechanism preventing query starvation, when the waiting time predominate the potential benefits from using the speculative query results.

D. Modifying Queries

In Section IV(A), it was said that the query test sets contained approximately 4% of modifying queries. Since the modifying query enters the Speculation Window until it reaches its head and is executed as a nonspeculative query, it endangers all speculative queries to be executed on invalid data. Just after the execution of a modifying query a validation process is executed for the queries marked with the Speculative State. The aim of that process is to save (positive validation) as many speculative results as possible from being deleted. For the execution of the whole query test set with the old strategy an approximate number of 40 speculative results (for 1000 user queries) were saved from being deleted and 22 had to be deleted due to negative validation. Experiments show, that the new strategy for the nonspeculative query execution doesn't interfere with the strategy for the modifying query handling presented in [6]. For the SW size 5 numbers of deleted and saved queries remains almost unchanged. With the growing size of the Speculation Window the number of deleted speculative results grows slowly to reach an approximate value of 36 queries for the SW size 15. The number of queries endangered with invalid data and saved from being deleted grows faster and reaches an approximate value of 130 for the biggest SW, which in general doesn't seem to have a negative influence on the speculative execution.

V. Conclusion

The paper describes a new strategy for the nonspeculative query execution in the speculative query execution support system called the Speculative Layer. We now allow to change the order of user queries execution, if the first query in the Speculation Window would have to be executed without the use of the speculative results. The first series of experimental results for the test database and three sets of 1000 input queries each, reduced the number of user queries executed without the use of the speculative results by approximately 50%, thus in general, almost 95% of user queries were executed with the use of at least one speculative query results. The proposed speculative support provides the query average execution time reduction up to 9 times for long running queries

TABLE I
EXECUTION RESULTS OBTAINED FOR DIFFERENT SIZES OF THE SPECULATION WINDOW

	Speculation Window Size										
	5	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Nojob reports	70	85	106	135	142	169	200	186	185	200	232
No Spec. Results Used	57	29	31	35	35	34	39	36	34	32	36
Total Num. of Executed Spec. Queries	1748	1717	1667	1594	1592	1539	1496	1480	1481	1445	1419
Avg. Num. of Spec. Queries Assignment	1.98	2.14	2.25	2.47	2.66	2.81	2.9	3.12	3.29	3.46	3.56
Num.of Nonspec. Queries executed with changed order	58	61	61	66	66	70	70	66	64	59	66
Num.of User Queries order change	132	180	226	274	300	335	372	383	406	373	402
Max delay in Nonspec. Query execution	4	5	6	7	8	9	11	11	12	13	13
Avg. delay in Nonspec. Query execution	2.59	2.95	3.70	4.15	4.55	4.78	5.46	5.80	6.34	6.32	6.09

(approximately 2 times for the whole test set). The execution order change was reported for approximately 58 queries, with the maximal execution delay of 4 queries (average delay 2.59). With the second series of experiments we have proved that the SW size equal 5 is a good choice. Changes in the SW size provided minor changes in the number of queries executed with the use of the speculative results. On the other hand, the average delay of a nonspeculative query execution, grew for the bigger SWs from 2.59 to 6.09.

Further work will focus on the development of an even more flexible user query execution strategy for a Speculation Window. A concept of a Variable Shift Speculation Window will be considered, which includes more than one user query executed as a nonspeculative query for each SW, depending on the available speculative results.

REFERENCES

[1] A. Estebanez, D.R.Llanos, A.Gonzales-Escribano, "A Survey on Thread-Level Speculation Techniques," *ACM Computing Surveys,* vol. 49(2), pp. 1-39, 2017, https://doi.org/10.1145/2938369

[2] A. Sasak-Okoń, "Speculative query execution in Relational databases with Graph Modelling," *in Proceedings of the FEDCSIS 2016*, ACSIS, Vol. 8., pp.1383-1387, 2016, https://doi.org/10.15439/2016F123

[3] A. Sasak-Okoń, M.Tudruj, "Graph-Based speculative Query Execution in Relational Databases," *in ISPDC 2017*, Innsbruck, Austria, IEEE Explore, https://doi.org/ 10.1109/ISPDC.2017.14

[4] A. Sasak-Okoń, M. Tudruj, "Graph-Based Speculative Query Execution for RDBMS," *in PPAM 2017*, LNCS, Vol. 10777, pp. 303-313, https://doi.org/ 10.1007/978-3-319-78024-5_27

[5] A. Sasak-Okoń, M. Tudruj, "Speculative Query Execution in RDBMS Bsed in Analysis of Query Stream Multigraphs," *in 24th IDEAS 2020*, Seoul, Korea, pp. 208-218, https://doi.org/10.1145/3410566.3410604

[6] A. Sasak-Okoń, "Modifying Queries Strategy for Graph-Based Speculative Query Execution for RDBMS," *in PPAM 2019*, LNCS, Vol. 12043, pp. 408-418, 2020, https://doi.org/10.1007/978-3-030-43229-4_35

[7] J. Silc, T. Ungerer, B. Robic, "Dynamic branch prediction and control speculation," *Int. Journal of High Performance Systems Arch.,* Vol. 1(1), pp.2-13, 2007, https://doi.org/10.1504/IJHPSA.2007.013287

[8] S. Pan, K. So, J. T. Rahmeh, "Improving the accuracy of dynamic branch prediction using branch correlation," *in Int. Conference on Architectural Support for Programming Languages and Operating Systems,* Boston, 1992, pp.76-84, https://doi.org/10.1145/143371.143490

[9] A. Moshovos, S. E. Breach, T. N. Vijaykumar, G. S. Sohi, " Dynamic Speculation and Synchronization of Data Dependences," *in 24th ISCA, ACM SIGARCH Computer Architecture News,* 1997, Vol.25(2), https://doi.org/10.1145/264107.264189

[10] N. Polyzotis, Y.Ioannidis, "Speculative query processing," *CIDR Conference Proceesings,* Asilomar, 2003, pp. 1-12,

[11] G. Barish, C.A. Knoblock, " Speculative Plan Execution for Information Gathering," *Artificial Inteligence,* 2008, vol. 172(4-5), pp. 413-453, https://doi.org/10.1016/j.artint.2007.08.002

[12] P.K. Reddy, M. Kitsuregawa, "Speculative locking Protocols to Improve Performance for Distributed Database Systems," *IEEE Transactions on Knowledge and Data Engineering*, 2004, Vol.16(2), p.154-169, https://doi.org/10.1109/TKDE.2004.1269595

[13] T. Ragunathan T, R.P. Krishna, "Improving the performance of Read-only Transactions through Asynchronous Speculation," *SpringSim Conference Proceedings*, Ottawa, 2008, p.467-474

[14] V. Hristidis, Y. Papakonstantinou, "Algorithms and Applications for answering Ranked Queries using Ranked Views," *VLDB Journal*, 2004, Vol.13(1), p.49-70.

[15] X.Ge, B.Yao, M.Guo, et al., "LSShare: an efficient multiple query optimization system in the cloud," *Distrib. Parallel Databases*, 2014, Vol.32(4), pp. 593-605, https://doi.org/10.1007/s10619-014-7150-1

[16] M.B.Chaudhari, S.W.Dietrich, "Detecting common subexpressions for multiple query optimization over loosely-coupled heterogeneous data sources," *Distrib. Parallel Databases*, 2016, Vol.34, pp.119-143, https://doi.org/10.1007/s10619-014-7166-6

[17] G.Preti, M.Lissandrini, D.Mottin, Y.Velegrakis, "Mining patterns in graphs with multiple weights," *Distributed and Parallel Databases, Special Issue on extending Database Technology*, 2019, pp.1-39, https://doi.org/10.1007/s10619-019-07259-w

[18] O.Goonetilleke, D.Koutra, K.Liao, T.Sellis, "On effective and efficient graph edge labeling," *Distributed and Parallel Databases*, 2019, Vol.37, pp.5-38, https://doi.org/10.1007/s10619-018-7234-4

[19] H.M. Faisal, M.A. Tariq, Atta-ur-Rahman, A. Alghamdi, N. Alowain, "A Query Matching Approach for Object Relational Databases Over Semantic Cache," *Chapter in Application of Decision Science in Business and Management*, 2020, https://doi.org/10.5772/intechopen.90004

[20] M. Ahmad, M. A. Qadir, M. Sanaullah, "Query Processing Over Relational Databases with Semantic Cache: A Survey," *2008 IEEE International Multitopic Conference, Karachi*, 2008, pp. 558-564, https://doi.org/10.1109/INMIC.2008.4777801.

[21] F.Wang, G. Agrawal, "Query Reuse Based Query Planning for Searches over the Deep Web," *Database and Expert Systems Applications. DEXA 2010*. LNCS, Vol 6262, 2010, https://doi.org/10.1007/978-3-642-15251-1_5

[22] P. Cybula, K. Subieta, "Query Optimization by Result Caching in the Stack-Based Approach," *Objects and Databases. ICOODB 2010*, LNCS, Vol.6348, 2010, https://doi.org/10.1007/978-3-642-16092-9_7

[23] TPC benchmarks, http://www.tpc.org/tpch/default.asp, 2020.