Database Index Balancing Strategy

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

Abstract— **Database index is a specific object, by which the data can be accessed effectively. Default relational database index is B+tree, which is balanced to ensure consistent and fast access by the traverse path. By using index, data retrieval process can benefit. Vice versa, other manipulation operations must apply all changes to the structure causing the performance drop. This paper deals with the data indexing techniques. It extends the already defined Notice list layer and segregates the balancing strategy into the separate process, by which the original transaction can be approved sooner. Thanks to that, Insert, Update and Delete operations are influenced only in a minimal manner, but the robustness of the indexing strategy is always ensured limiting the whole data block set scanning necessity. It proposes own architecture concerning to the undefined values. They are placed in the main index. It the row migration caused by the Update operation is present, automatic management removes such limitation to ensure optimal layout of the index.**

I. INTRODUCTION

Relational database systems are defined by the database itself and instance formed by the memory structures and background processes managing the whole ecosystem. Users, developers and administrators cannot directly deal with the data, all operations are done and supervised by the instance. The aim of the optimization and main performance aspect is just the process of the data transfer from the database to the instance to evaluate and build resultset and vice versa, individual changes, new states and corrections are operated in the memory consecutively stored in the database [6]. Thus, the efficiency of the data transfer is inevitable to be highlighted and covered, whereas database systems form the core part of the whole information technology. Intelligent information systems need to manage and process not only current valid data, but the whole evolving data set over the time is highlighted [11], [12], [14]. Conventional database is characterized by treating only current valid data, any change forces the system to replace original state by the new version. Thus, historical data cannot be managed later, at all, although historical states can be temporarily obtained by the transaction logs to ensure consistency, isolation and durability aspects.

The aim of this paper is to cover the process of the data transfer between the database and instance and ultimately all the way between the system and the user. It deals with the indexes as optimized data locator module, migrated rows as the result of the Update operation, by which the new version does not fit the original position. The second part of the paper proposes own architecture dealing with NULL values directly in the index. Thirdly, new balancing method operations are introduced. Thanks to that, index itlesf is not degraded over the time.

The proposed paper is defined as follows. Section 2 deals with current trends and techniques in relational databases. Section 3 deals with the limitations. Section 4 covers the own proposed techniques to optimize data transfer using the index. Section 5 offers computational study.

II. STATE OF THE ART

Data are stored in the database, physically organized in the data segments (data definition) and extents (set of physical data blocks) [3], [21]. Extent is the allocation unit; blocks are not created individually to ensure the performance. Individual blocks and extents are interconnected via the linear linked list, starting by the object definition itself, up to the High Water Mark (HWM) symbol pointing to the last associated block to the data structure, either table or index. Data are stored in the data blocks. Generally, they are distributed randomly, system automatically finds the available block [1], where the data portion can be placed. After the Update and Delete operations, blocks can be fragmented ending in managing partially or totally free space in the blocks. Thus, as the consequence, some blocks can be totally empty, caused either by the flushing the block, or as the result of new extent association. If the user wants to get relevant data from the database, they must be located in the first phase. For dealing with the data, two principles can be identified. The easiest, but the most demanding solution is based on the scanning whole data block set sequentially. Each block is transferred from the database into the memory Buffer cache for the evaluation. There, the block is parsed locating data rows, which are consecutively evaluated, whether the conditions of the original statement are met or not. The most demanding operation is just the I/O operation during the loading process. Note, that the block can be even empty with no relevant data inside.

The second evaluation stream principle is delimited by the index usage. Database index is a specific object stored in the database, by which the relevant data portion can be easily located in the database. It can be formed by various architectures, like B+tree [4], bitmap [5], hash [10], etc. The most often used relational index is a B+tree defined by the root node, internal nodes and leaf layer nodes, which consist of the pointers to the physical database layer – ROWID. Address values ROWID require 10 bytes and contain the

information of the data file, data block and position of the block inside the block itself [4], [8]. By using index, traverse path ensures logaritmic complexity. By raising ROWID value, either unique row can be identified or the range of adresses can be used. In the second phase, physical ROWID is parsed and particular block is loaded to the memory Buffer cache for the consecutive processing and evaluation. As evident, by using the index, only relevant data blocks are loaded into the memory. As the result, the processing time, I/O operation demands are lowered, whereas the amount of data to be processed and evaluated is significantly lowered. If the amount of data to be treated is high (based on the experiments, the border is approximately 20% of the total data amount [2], [5]), whole table is scanned sequentially. If files are distributed across multiple physical discs operated by the several interfaces, loading and processing can be done in parallel. One way or another, it is evident, that the data amount to be treated is really significantly lowered. Data service robustness is a core element dealing with the data ensuring performance [16], which must be reflected to any data structure perspective [22].

III. NULL VALUE AND MIGRATED ROWS MANAGEMENT

In principle, there are two core methods dealing with the data – Index Unique Scan and Index Range Scan [4], [15], [22], which differentiate the result set data amount. If the condition is based on the unique constraint, result set consists no more than one row, thus if the ROWID is located in the index, processing in this stage can be ended. If the condition can result in producing multiple rows, particular data ROWID pointer is located in the leaf layer, the next nodes are scanned for the condition as well. In comparison with standard B-tree, it uses the benefit, that the leaf layer nodes are interconnected. The result of such index processing is a list ROWIDs, which point to the blocks with the relevant data. Such blocks are loaded into the memory by invoking Table Access by Index Rowid (TAIR method) [15], [18].

Similarly, if the data are to be updated, these must be located and loaded into the memory for the replacing operation. If the data model evolve over the time, problem can be even deeper covering all integrity constraints and rules [7].

As stated, problem is just the efficiency of the data location inside the index. The most often used B+tree index is balanced tree ensuring, that the traverse path length from the root element into any leaf node is always the same. By reaching this prerequisite, optmizer can pre-calculate the estimated costs of the processing. Generally, three problem areas regarding the index structure can be identified influencing the global performance.

The first problem is associated with the data block positions, which can be incorrect reflecting the situation, that the original block is not suitable for the updated tuple. If the size of row after the change does not fit the original block, database processor must look for another suitable block, which can hold particular object. It means, that the ROWID pointer of the index adresses the original block, where pointer to the another block is present. As a consequence, during the data block access, two blocks must be loaded – original block, to which the index ROWID points and the block, where the data actually resist. In general, several migrated rows can be present, with various depth, as well. Thus, not only two blocks need to be loaded and scanned. Whereas the index is not notified, performance of the system based on the index access is still worse and worse. Existing solution is based on the index rebuild option [9], [13], which is not, however, conventient due to multiple factors. First of all, it requires many system sources to perform such operation. If the offline mode is used [4], original index is dropped, followed by the new index creation. Thus, during the rebuild process, table must be scanned sequentially resulting in really poor performance produced to the user. Vice versa, if the online mode of the index rebuild operation is used, new index is created, while the original still exists in the system. Therefore, additional storage demands are required, as well as many system resources consumption. Our proposed solution uses notification layer, which does not require whole index rebuild operation, just the changes reflecting migrated rows are applied.

The second problem deals with undefined values, which are not covered in the database index. The reason is based on the relational algebra, which cannot compare NULL values mathematically forming the 3 valued-logic. Introducing NULL values to the evaluation results in the necessity to maintain such values in a specific manner. Any mathematical operation based on the NULL value produces NULL as the result [17]. Therefore, index traverse operations based on the operand smaller (<) and larger (>) cannot be applied to the NULLs. Similarly, condition based on the equality cannot be used, as well. Although it can be solved by the NULL management module introduced in [15], it has several limitations regarding the performance, whereas particular values are not part of the index and must be loaded separately on demand. In this paper, we propose own solution for loading and managing NULL values inside the index, mostly with emphasis on the memory efficiency. In spatio-temporal environment, the whole time spectrum is modelled, even with emphasis on the data corrections, so many undefined values and fragmentations can be present [15], [17], [19].

The third problem covered by this paper is based on the balancing process. Database index is formed by the B+tree index tree, which is always balanced, thus the traverse path length is always the same. The balancing activity is done during any data change, either directly or postponed to the end of the transaction, thus after reaching commit (transaction approval), each change is index reflected and the each index tree is balanced. In [15], index balancing operation has been extracted into the separate transaction process, thus the main transaction can be processed and approved sooner. If multiple data rows are updated and new protions are loaded, it can provide significant benefit. Based on the study defined in [15], for the sensor based network, the processing time costs are lowered to 73%. Physically, any change was stored in the

Notice list, by which the introduced background process Index_balancer was notified. As a consequence, data retrieval process needs to check the index itsef, as well as the Notice list, which can be demanding, if the data amount is continuously changing. If the environment is shifted into the temporal sphere [10], [20], which is nowadays very widespread, whereas it is necessary to evaluate not only current valid data, but the whole data evolution over the time, the problem is even deeper and the pressure to ensure reliability and performance is sharpened. The solution is based on adding data directly into the index irrespective of the balancing. After the change, Balancer background process is activated. In that case, data retrieval does not need to cover two structures, whereas it is ensured, that relevant data are always present in the index.

Next section can be divided into three parts, the first deals with the migrated rows. In the second part, we propose solution for dealing with NULL values. The third part covers the balancing operations by introducting new background processes and specific architecture.

IV. SOLUTION – BALANCER LAYER, INDEX STRUCTURE EXTENSION

A: Migration

To ensure performance efficiency of the whole system, it is inevitable to cover all the data inside the index, to propose robust architecture reflecting the attribute list and order of individual attributes of function results inside. The order of the elements is important, covered by [4], [5]. If the attribute order fits the Select statement, database optimizer selects between Unique and Range Scan. Specific solution provides Full Scan methods, by which the index is either scanned sequentially for the evaluation by assuming, that the size of the index is always smaller in comparison with the whole table. Fast Full Scan method is suitable, if the order of elements is not correct for the Select statement, however, the index itself holds all data attributes necessary for the evaluation. The core problem is the data row migration. Although the system idenfities relevant data block, several ones must be loaded instead to locate particular row. Fig. 1 shows the architecture.

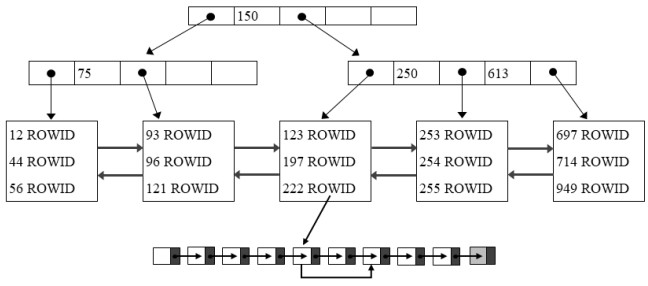

Fig. 1. Migrated row management [4]

There is an index locating the row by using ROWID. The direct data block is stored in the database necessary to be loaded. Block 1 is transferred to the memory Buffer cache for the evaluation. There is, however, another pointer, thus the Block 2 is loaded resulting in multiple loading operations. As evident, Block 1 does not need to be loaded. If the index holded new ROWID, just the Block 2 would be loaded instead. There is, however, no pointer from the block to the index, thus index canot be notified without the rebuild opeartion, which, however, reconstructs the whole index. To solve that limitation, we introduce another module. Any change resulting in migrated row necessity notifies this layer. Then, the Balancer background process checks the prerequisite in the Migration module and applies changes to the index. Whereas the balancing is done outside the main operations, they are not absolutely influenced in a negative manner. Proposed architecture is shown in Fig. 2. Several indexes can be present in the system. For each data row in the Migration module, there is a check box list indicating, whether the identified migrated row has been applied to the index or not. Naturally, it is checked by the Balancer background process and if the index holds all table data portion, such row change would be always present. If the index does not manage NULL values, such data pointer does not need to be there. In that case, however, such index is marked as already applied such change.

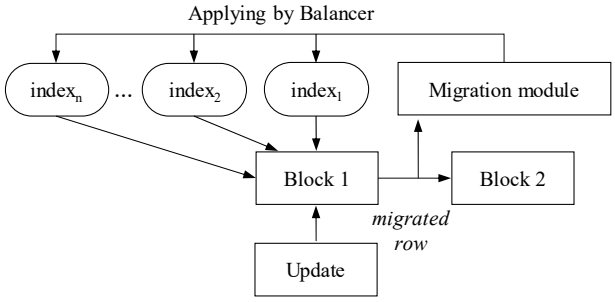

Fig. 2. Migration module

B: NULL value management

The second part of the proposed solution is based on the NULL value management. As stated, undefined values are not directly part of the index. In [15], NULL management module has been proposed. For the purposes of the performance, undefined values cannot be randomly distributed in the container covering undefinition. In [15], it has been sorted based the time validity or spatial positions. In this paper, we propose new architecture, by which each undefined data portion is delimited by the transaction origin (its unique identifier) and time position of the transaction approval. If using the temporal database concept, direct predecessor and descendent is referenced, as well, to create complex evolution strategy. Optionally, the amount of the predecessors can be set, by default, value 1 is set. Borders specified by the time frame can be used, as well. Thanks to that, as undefined values can be identified bar more easily, whereas particular transaction data are always present.

Limitation of the already proposed solution in this paper is just the overall architecture. NULL values are located outside the main index resulting in the necessity to scan to structures. Main index hold only correctly specified values, undefined values are present in the separate database segment. Thus, such blocks must be loaded into the memory, similarly to the

index itself. As a consequence, the number of blocks to be loaded is increased – reflect also the aspect of the fragmentation of the blocks, which can be present. Therefore, this paper evaluates the benefit of two structures interconnection forming the global index_null structure. Let create an index based on the attribute A, B, C of the table T. The command to create such index is following (the name of the index is "ind"):

Create index ind on T(A,B,C);

In the above case, undefined values are not managed, at all. The existing solution to cover all data is to transform the undefined values using the function call, like NVL:

Create index ind on T(nvl(A, 'x'), nvl(B, 'x'), nvl(C, 'x'));

The limitation of the function call is based on the extended processing demands, but mostly size demands. NULL value itself does not require extra storage, after the transformation, at least 1 byte is inevitable for each data row, based on the data type. Generally, it can require many bytes.

To create NULL management module externally, following command extenstion can be used. Note the keyword NULL_module forcing the system to create extra module, outside the main index holding NULL values. As stated, by default, undefined values inside the NULL management module are sorted based on the time perspective or position (association to the region) respectively.

Create index ind on T(A,B,C NULL_MODULE);

In the above case, NULL values are stored externally in the separate segment for each associated index. There are no extra demands based on the tuple amount, just the NULL_MODULE is added as a structure.

The third approach dealing with the NULL values inside the index is based on the internal representation. In comparison with the function call, data are transcoded internally on demand, thus no additional size demands are present. There is just the condition expressing the positioning principles – how the data are organized inside the index, just with emphasis on the undefined values. By default, particular undefined values are stored internally, however, there is also option to manage them separately to the mainly defined data. The main difference between the already stated approaches is based on the fact, that in any case, all indexed data are part of just one segment, either stored internally or externally. Reflecting this fact, the proceduce can significantly benefit, whereas the loading process does not need to locate to segments, block from the outer position is always the same, so the I/O loading process is far easier and straightforward.

The syntax of the solution consists of the three element groups – list of attributes, location of the undefined values (external or internal) and the last module expresses the condition set, which is applied during the data location. Such conditions replace the original mathematical approach considereing NULLs. Compared to the function transformation, this proposed solution stores NULLs value physically as NULLs (with no additional demands), for the evaluation, transformation is used, either to locate existing data tuples, but it is used for the placing new rows into the structure. Architecture of the solution is shown in Fig. 3.

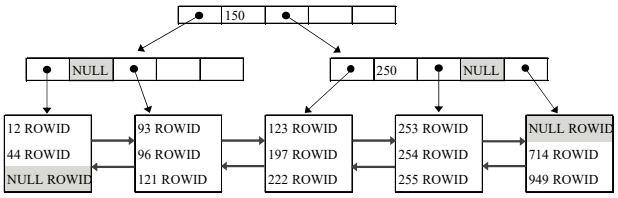

Fig. 3. NULL inside the index

Technically, we can see NULL value directly as part of the index, but during the data retrieval, optimizer automatically shiftes new condition reflecting the possibility to position undefined value inside the main index structure.

Create index ind on T(A,B,C,
{internal | external},
condition_set1 {,condition_set2, …});

C: Balancing

The last part to be handled in this paper is reflected by the index balancing. In principle, two opposite categories influenced by the index structures can be identified. The first group benefits from the index existence and uses it to locate data easily, just by traversing index and locate pointer directly shifting the processing to the relevant block. This group is formed by the data retrieval operation – Select and can partially cover locating operations during other operations (Insert, Update and Delete). This category handles the integrity constraint rules, too. The opposite group is covered by the operations; which demands are increased. Processing time is extended slowing down the performance. Additional costs are associated with the index operations, whereas each change must be applied there. Our proposed solution refers to the solution rule based on the post-indexing layer. Notice list was the core element, which registers any change applied to the index firing the trigger operation to balance the structure. Originally, balancing was done in the two phases. New data portion was located in the Notice list in the first phase, followed by applying change into the index structure directly. Each index has its own Notice list module. Reflecting the practical usage of the existing solution, two bottleneck limitations can be identified. If any change was applied to the data, such change vector would be necessary to place to each element of the Notice list layer set. Thus, if the table consists of 10 indexes, the change vector had to be copied ten times, always with the same information. Secondly, it cannot be ensured, that the data change is applied to each index at the same time. Therefore, particular index usage was strongly delimited by the amount of change vectors waiting to be applied in the index. Database optimizer, however, cannot

calculate the estimated additional costs reflecting the Notice list consequencing in inproper decision. Although the index is selected to be suitable for the query – it can have the optimal structure based on the condition set and selected attributes, the global performance does not need to be ensured. The main reason reflects the index relevance and applied changes. If the update stream is too high, it can naturally happen, that most of the data are paced in the change vectors of the Notice list, which is modelled by the linked list, thus the benefit of the original B+tree is erased, if most of the required data are not in the index, but in the pending structure for processing by the Index_balancing process. The problem is therefore clear, it is necessary to create new robust solution process for the balancing. The solution presented in this paper is based on the external background process managing the balancing operation. Thus, it is not ensured, that the traverse path from the root to any leaf node is always the same, but any node can be directly located inside the structure. The rule, that reference values with lower key are in the left part, while higher key values can be located in the right part is always met. Example of the tree (for the explanation reasons, it is modelled by the binary tree) is shown in Fig. 4, 5 and 6. Let original tree consist of the 6 elements – Fig. 4. Now, the tree is suitably balanced. For the insert operation, let's assume to add four new elements into the index (values 1, 2, 6 and 10) – Fig. 5. After adding values 6 and 10, right part of the tree is extended. Value 2 is connected to the value 3, value 1 is connected to newly inserted node 2. As shown in fig. 5, particular tree is not balanced. All new row references are added to the index. In this moment, original transaction can be accepted and approved. Fig. 6 shows the solution after the balancing operation. It is evident, that the depth of the left part of the tree is lowered from the value 4 to 3, by rotating the internal node 3 to the leaf. For the result, decision internval element would be node 2, instead of 3, done by the right side rotation. By using this external technique, change operations can be ended successfully significantly sooner, on the other hand data retrieval process is not optimal from the index structure point of view, whereas it is only partially balanced anytime. On the other hand, Balancer background process is triggered as soon as the balancing is necessary, thus it is done almost immediate. The processing and global performance benefits from the fact, that multiple balance operations can be done at once, in comparison with the balancing necessity for each transaction operation. Note, also, that the balnacing can be spread across multiple transactions.

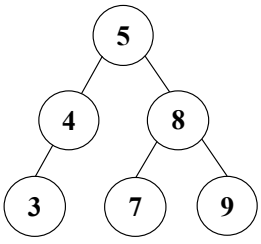

Fig. 4. Original index structure

Next paragraph deals with the complex architecture of the solution solving all mentioned performance limitations. Fig. 7 shows the architecture.

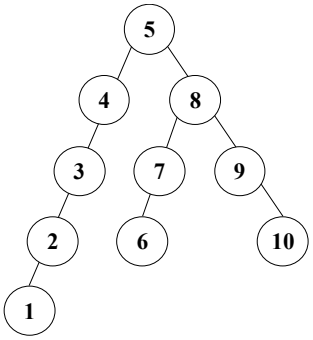

Fig. 5. Index structure after the Update operation (no balancing)

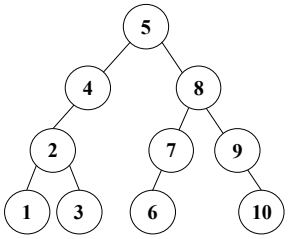

Fig. 6. Balancing outside the main transaction - results

D: Complex architecture

To cover the efficiency of the proposed solutions, complex architecture has been developed covering all dealt problems and limitations. Core part is formed by the optimizer, to which the query is shiften to identify the most beneficial data access path. Optimizer decision is based on the query definition, its parsing and database statistics, which must be up to date to reach optimalized performance [13]. Autoindexing has been defined by the DBS Oracle, by which the definition of the index set can be enhanced automatically by adding and removing indexes from the system based on the current queries, optimizing strategy and global performance. It uses extended statistics and virtual indexes to evaluate costs and benefits of the particular index. Left part of the architecture shot deals with the balancing. Note, that each change is directly applied to the index, Index Balancing Requests module just references the changes, so the balancer can easily identify subject of interest. Two additional modules can be located there. NULL Manager background process deals with the undefined value transaformation internally to propose categorization and location by removing the impact of direct mathematical comparison limitation. Migration module is another extension listing data block shiftments operated by the Migration manager background process. The whole architecture is in Fig. 7.

Next section offers the performance evaluation of the proposed solutions, categorized based on the three levels defined in section 4. Namely, the first part covers the migrating row impacts, the second part deals with the NULL

value implementation inside the index structure and the last experiment evaluation manages the balancing processes.

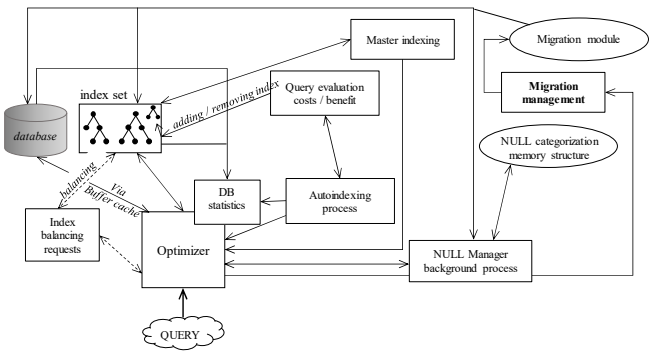

Fig. 7. Complex archtiecture – Index balancing, NULL management and Migration module extension

V. COMPUTATIONAL STUDY

Performance characteristics have been obtained by using the Oracle 19c database system (Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 – Production). Parameters of used computer are processor: Intel Xeon E5620; 2,4GHz (8 cores), operation memory: 16GB, HDD: 500GB. For the evaluation, a table containing 10 attributes originated from the sensors were used, delimited by the composite primary key consisting of two attributes. The table contained 1 million rows, 10% of them contained undefined values. Two index structures were defined, one implicit covered by the primary key, the second index extends the primary key by covering attribute, which can hold NULL value. The Select statement was evaluated covering all attribute values. The condition limited the amount of data to 10% of the whole data set. One percent of rows was stored in a migrated blocks. Data set management and structure was introduced in [15].

The reference model for the evaluation and comparison was **MODEL 0** defined by the two *B+tree* indexes (primary key and user-defined) with no undefined values special support. There is no support for the NULL value handling, nor external balancing. Thus, in the results, this model reaches 100% of the processing costs and size demands. To evaluate the the performance benefits, size demands, and processing time of the Select statement is analyzed in this paper. Three experiments can be identified in the main part of this section, afterwards, the complex architecture is created and evaluated, reflecting the section 4. Experiment 1 deals with the proposed solution for data migrating impact removal, by introducing Migration module. Experiment 2 covers the undefined values. It compares already existing systems with the internal transformation, so the undefined values can be stored directly in the main index. In Experiment 3, balancing demands and benefits are evaluated.

Experiment 1

The referential model (MODEL 0) for this study does not use any addional layer for providing migrated row management. Thus, the efficiency of the index structure is lowered over the time. As opposite, new index for such timepoint is created, so its structure is optimal for that moment (MODEL 1). Both these models form the borders reflecting the performance. Our proposed solution is defined by the MODEL 2 mark. As evident, performance is almost the same in comparison with MODEL 2, the difference is 1,3% for the data management (Insert, Update, Delete and Select statement). It must be, however, stated, that reaching optimal performance delimited by the MODEL 1 forces the system to rebuild indexes regularly. In dynamic systems, it can be even so often that the original index does not even have time to build completely and is no longer up to date. As mentioned, the additional costs are at the level of 1.3% with 1% fragmentation caused by migrated rows (experimented 10 times, values express the average value), but the index is still usable and regularly directly optimized in the soil structure. Fig. 8 shows the reached results. 1 million of Update operations were performed, 1% of them required additional block space, which cannot be provided directly. Thus, it required new storage allocation, multiple blocks were interconnected. Values in fig. 8 are expressed in percentage reflecting additional costs of the processing time. Optimal solution is provided by the MODEL 1.

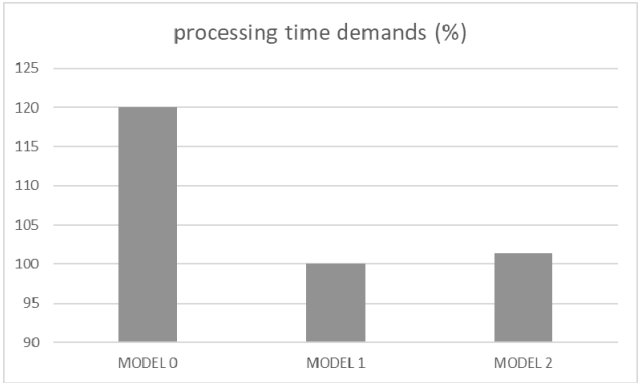

Fig. 8. Migrated row management

Experiment 2

NULL management can be processed either externally by using NULL management module (MODEL 3), externally using the proposed solution in this paper (MODEL 4) or internally by the transformation od demand – data are stored as NULL, however, for the evaluation, autotransformation is done to user availability of the mathematical comparison – MODEL 5. Fig. 9 shows the reached results. Reference model is MODEL 0, in which no undefined values can be present. For the evaluation, 10% of the data are selected, of which 10% are undefined. Based on the study, whereas MODEL 0 cannot hold undefined value, whole block set associated with the table must be scanned sequentially, reaching 100%.

Whereas 10% of data are to be processed, in optimal environment, it should be done in 10% of the total demands of the table. Such results cannot, be, however, reached, whereas the index access must be identified, particular index node blocks need to be loaded into the memory Buffer cache, if they are not present there. NULL management module (MODEL 3) requires 15,7% of the processing time, which is mostly enhanced by the two separate segments, which must be loaded and evaluated. Loading is a sequential process generally,

evaluation can be done in parallel in the memory Buffer cache scanning. MODEL 4 uses autotransformation and stores data in the same segment, although the delimitation between these strucures can be be easily identified. The total demands are 14,5%, there. MODEL 5 provides the best solution by storing undefined data internally. In comparison with function based index, the total improvement can reach up to 50% due to additional disc storage capacity, as well as the necessity to revert the function results to the original form for the result set produced to the user.

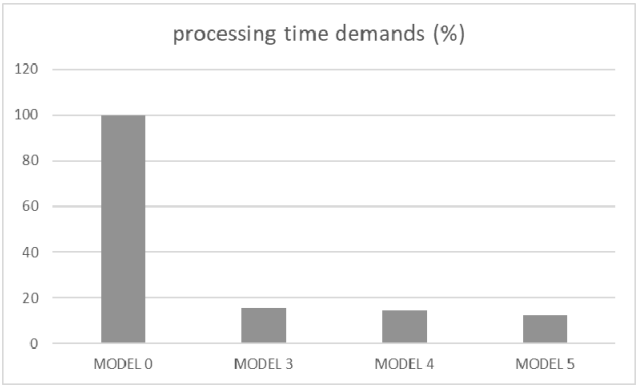

Fig. 9. Undefined value management results

Experiment 3

Balancing operation is an inevitable part of the index management. Without such activity, original index can degrade to form linear linked list by shifting the complexity from O(log(n)) to O(n). The original perspective of the B+tree indexing is based on the fact, that individual nodes are always balanced. In our proposed solution, index balancing operation is extracted into the separate transaction. For the purposes of this paper, it is compared to the two existing approaches MODEL 0 is referential, so the balancing operation is done directly in the main transaction ensuring always balanced structure after the transaction approval. MODEL 6 categorizes the balancing, it is operated outside the main transaction, howver, particular changes are noted in form of change vectors stored in the Notice list. Thanks to that, during the data retrieval, index is scanned in the first phase, followed by the Notice list traversion, where the data are array formed generally. MODEL 7 uses the proposed architecture, in which new node is placed in the leaf layer irrespective of the balancing, such node is marked to be balanced and introduced Balancer background process is notified to start balancing. Fig. 10 shows the results managing data retrieval process by using index. Index range scan is used, thus it provides optimal order of attributes, there. MODEL 0 requires 100%, MODEL 6 requires additional 14,2%. MODEL 7 places data directly to the index structure and balancing is then done separately. It requires additional 11,3%. Note, that the transaction approval process is lowered in MODEL 6 and MODEL 7 up to 30%.

VI. Conclusions

Performance of the data management in the database is a crucial element of almost any information system. It is evident, that the amount of the data to be processed and evaluated is still rising very rapidly. Effective methods for dealing with the data, from the access perspective, are really important. In this paper, three core problems were discussed. The first one is associated with the data row migration. After the update operation, new state does not need to fit the original place inside the block, so it must be shifted into different position in another block. As the result, migrated row is created, however, the index set still points the the first – original block. Thus, several blocks need to be loaded into the memory for the evaluation. Our proposed solution is based in additional specific layer notifying index balancer to apply the changes. Thanks to that, impact of the data migration is limited in time. From this point of view, the index does not lose efficiency over time.

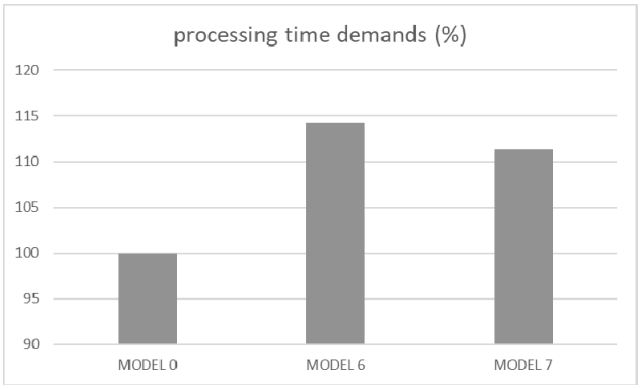

Fig. 10. Balancing operation - results

The second problem dealt is based on undefined values. Many times, data reliability cannot be ensured, resulting in the necessity to store and evaluate undefined values, marked as NULL. Such values, however, cannot be mathematically compared. Proposed solution defines transformation modules managed internally, so any value can be part of the index.

The third element covered by this paper is reflected by the index balancing. The processing itself is extracted and done in the separate transaction operated by the Balancer background process. Thanks to the proposed architecture, any data change can be approved sooner by applying the change into the index leaf layer, however the balancing is not done directly. The depth of the traverse path is changed and internal background processes are notified to start activity. It uses the fact, that multiple change operations can be routed to the same balancing operation, thus, finally, the costs of the balancing is lowered.

In the future, our emphasis will be on the complex index supervision layer, by which the index set can be optimized. It will use index in index strategy, so the Where condition of the Select statement can be divided into multiple tasks evaluated in parallel, by which the processing can end sooner.

Acknowledgment

This publication was realized with support of the Operational Programme Integrated Infrastructure in frame of the project: Intelligent systems for UAV real-time operation

and data processing, code ITMS2014+: 313011V422 and co-financed by the Europen Regional Development Found.

REFERENCES

[1] Abdalla, H. I.: A synchronized design technique for efficient data distribution, Computers in Human Behavior, Volume 30, 2014, pp. 427-435

[2] Behounek, L., Novák, V.: Towards Fuzzy Patrial Logic. In 2015 IEEE Internal Symposium on Multiple-Valued Logic, 2015.

[3] Bottoni, P., Ceriani, M.: Using blocks to get more blocks: Exploring linked data through integration of queries and result sets in block programming, IEEE Blocks and Beyond Workshop (Blocks and Beyond), 2015.

[4] Bryla, B.: Oracle Database 12c The Complete Reference, Oracle Press, 2013, ISBN – 978-0071801751

[5] [5] Burleson, D. K.: Oracle High-Performance SQL Tuning, Oracle Press, 2001, ISBN - 9780072190588

[6] Ceresnak, R., Matiasko, K., Dudas, A.: Influencing migration processes by real-time data. In Proceedings of the 28th conference of Open innovations association FRUCT

[7] Delplanque, J., Etien, A., Anquetil, N., Auverlot, O.: Relational database schema evolution: An industrial case study, IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Spain, 2018, pp. 635-644

[8] Eisa, I., Salem, R., Abdelkader, H.: A fragmentation algorithm for storage management in cloud database environment, Proceedings of ICCES 2017 12th International Conference on Computer Engineering and Systems, Egypt, 2018

[9] Feng, J., Li, G., Wang, J.: Finding Top-k Answers in Keyword Search over Relational Databases Using Tuple Units, IEEE Transactions on Knowledge and Data Engineering (Volume: 23, Issue: 12, Dec. 2011) , 2011.

[10] Honishi, T., Satoh, T., Inoue, U.: An index structure for parallel database processing, Second International Workshop on Research Issues on Data Engineering: Transaction and Query Processing, 1992.

[11] Janáček, J., Kvet, M. (2016). Sequential approximate approach to the p-median problem. In Computers & Industrial Engineering 94 (2016), Elsevier, ISSN 0360-8352, pp. 83-92.

[12] Jánošíková, Ľ., Kvet, M., Jankovič, P., Gábrišová, L. (2019). An optimization and simulation approach to emergency stations relocation. In Central European Journal of Operations Research, ISSN 1435-246X, Roč.27, č.3 (2019), pp. 737-758.

[13] Kriegel, H., Kunath, P., Pfeifle, M., Renz, M.: Acceleration of relational index structures based on statistics, 15th International Conference on Scientific and Statistical Database Management, 2003

[14] Kvet, M. (2019). Complexity and Scenario Robust Service System Design. In Information and Digital Technologies 2019: conference proceedings, Žilina, 2019, ISBN 978-1-7281-1400-2, pp. 271-274.

[15] Kvet, M.: Managing, locating and evaluating undefined values in relational databases. 2020

[16] Kvet, M.: Complexity and Scenario Robust Service System Design. In Information and Digital Technologies 2019: conference proceedings, Žilina, 2019, ISBN 978-1-7281-1400-2, pp. 271-274.

[17] Lien, Y.: Multivalued Dependencies With Null Values In Relational Data Bases. In Fifth International Conference on Very Large Data Base, 1979.

[18] Mirza, G.: Null Value Conflict: Formal Definition and Resolution, 13th International Conference on Frontiers of Information Technology (FIT), 2015.

[19] Moreira, J., Duarte, J., Dias, P.: Modeling and representing real-world spatio-temporal data in databases, Leibniz International Proceedings in Informatics, LIPIcs, Volume 142, 2019

[20] 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.

[21] Tilgner, M., Ishida, M., Yamaguchi, T.: Recursive block structured data compression, Proceedings DCC '97. Data Compression Conference, 1997.

[22] Vinayakumar, R. Soman, K., Menon, P.: DB-Learn: Studying Relational Algebra Concepts by Snapping Blocks, International Conference on Computing, Communication and Networking Technologies, ICCCNT 2018, India, 2018