International Journal of Engineering Trends and Technology
ISSN: 2231–5381 / https://doi.org/10.14445/22315381/IJETT-V71I11P209
Volume 71 Issue 11, 90-99, November 2023
© 2023 Seventh Sense Research Group®

Review Article

Database Tuning from Relational Database to Big Data

Bery Leouro MBAIOSSOUM[1*], Ladjel BELLATRECHE[2], Narkoy BATOUMA[1], Ahmat Mahamat DAOUDA[1]

[1]*Faculty of Exact and Applied Sciences (F.S.E.A), University of N'Djamena, NDjamena Chad.*
[2]*Ecole Nationale Supérieure de Mécanique et d'Aérotechnique, Chasseneuil-du-Poitou (France).*

**Corresponding Author : bery.mbaiossoum@gmail.com*

Received: 09 December 2022 Revised: 29 March 2023 Accepted: 10 October 2023 Published: 04 November 2023

Abstract *- The objective of this work is to present the database tuning from relational databases to Big Data. It revisits the tools of the physical design. It examines their applicability to the main types of databases, in particular legacy (hierarchical or network), relational, object-oriented and NoSQL databases and Big Data. A literary review is done on database tuning tools to examine how to bring them closer to the DB lifecycle. It is noted that modern physical design techniques consider all phases of the DB lifecycle. Database tuning has evolved as new database types emerge. There is a vertical evolution of database tuning with the addition of new phases when a new database type appears and a horizontal evolution resulting in the enrichment of each phase of the DB lifecycle by considering new tools. This phenomenon with the Bigata data results vertically with multiple types of Big Data schemas and horizontally with the advent of the map-reduce technique. The database administrator has to consider these evolutions in the database tuning work.*

Keywords *- Databases tuning, Queries optimization, Big Data, Databases evolution, Databases lifecycle, NoSQL.*

1. Introduction

Tuning advanced databases that handle large volumes of data, such as data warehouses [1, 2], is an important issue for the database community. The concern to quickly have the result of a query executed on a Database (DB) remains constant for all users. The DB administrator is responsible for implementing the necessary data structures to guarantee the database performance to users.

This aspect of the administrator's job is a matter of physical design called database tuning. Physical design is defined as the phase of the DB lifecycle where the implementation of structures to ensure the efficiency of the DB is done. The goal of physical design is query optimization so that query results are very quickly obtained [1].

In the paper [3] titled Self-Tuning Database Systems: A Decade of Progress (10 Year Best Paper Award at the Very Large Databases conference, 2007 edition), Surajit Chaudhuri indicates that the first generation of relational execution engines was relatively simple, focused on OLTP (online transaction processing), making index selection less problematic. The importance of physical design has been magnified as query optimizers become sophisticated to deal with complex decision-support queries [3]. This amplification is due to the following characteristics linked to advanced databases:

- The complexity of the database schema, where the tables do not have the same characteristics. Take the example of a diagram of a relational data warehouse and there are two types of tables: the normalized fact table containing a large number of instances and smaller dimension tables, often de-normalized to minimize the number of joins required to evaluate a query,
- The complexity of queries, which increasingly require expensive operations such as join and aggregation;
- The volume of data: more and more data in DB becomes important, especially with DB applications linked to the internet and social media; a lot of the data is saved in DB;
- The requirements of decision-makers on the response time of queries and;
- The diversity of optimization structures.

Another diversity is the use of these optimization structures. Some are applied during database creation, like horizontal fragmentation and others during database exploitation, like materialized views. This complicates the process of selecting and using some sensitive structures. Among the physical design tools, some apply to several types of DB, and others are less so.

The objective of this work is to revisit the physical design tools and examine their applicability to the main types of databases, particularly relational, object-oriented and NoSQL databases since database communities are interested in physical design topics, as testified in the paper of Eslami et al. [2] and Ding et al. [4].

This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

Query optimization in the context of databases has occupied an important place across the different generations of databases: traditional databases, XML databases, decision databases, statistical and scientific databases and semantic databases. Indeed, database applications are always looking for more efficient query processing time. Query optimization, therefore, consists of rewriting the query execution tree to choose the most efficient execution plan [3, 4]. Before executing a query, parsing is done for syntax checking and translation into algebraic operations. This analysis produces a tree of operations to be executed. However, it is possible to transform this tree to obtain other equivalents, which offer different means to get the same result. These trees are called execution plans [5-7]. The role of the optimization engine (also called optimizer) is to generate the different execution plans and choose the best one. Moreover, it is the latter that will be executed.

Several works have been carried out to make query optimizers more efficient. In the relational database, various optimization algorithms have been proposed [4-7]. The guiding idea is to move the selections down the query tree as much as possible. Palermo [8] was the first to propose this optimization strategy. Gotlieb [9] proposed an alternative to calculate joins. Yao [10] analyzed alternatives for select-project-join type queries. The QUEL decomposition algorithm was proposed by Yang [11], and the SEQUEL optimization by Astrahan et al. [12] and Selinger et al. [13]. Optimization of conjunctive queries has been the subject of the work of Chandra and Merlin [14]. This work was extended by considering multi-valued functional dependencies by Aho et al. [15] and by considering the union operator by Sagiv and Yannakakib [16].

In object-oriented DB, optimization techniques have been proposed, especially indexing [17-20], algebraic rewriting techniques [21, 22, 18], path expressions [15], etc. Kim et al. [17] have introduced and evaluated the path index. Kemper et al. [18] have proposed a generic optimizer called GOM, which uses path indexes in the form of support relations and a rewrite rule-based optimizer. Lanzelotte et al. [23] proposed an extensible optimizer allowing the complete taking into account new data types, with operators, rules, etc. The physical design has evolved and gradually addressed the entire DB lifecycle. Indeed, the query optimization process gradually took into account parameters from the phases of the database design lifecycle (conceptual design, logical design, physical design and deployment (Fig.1)) and the query language used by the target Database Manager System (DBMS). There are two main generations of query optimizers: optimizations directed by certain lifecycle phases (ODCP) and optimizations directed by all phases (ODAP).

- ODCP optimizers are mainly based on studying the algebraic properties of the query language defined on the logical model of the database [6]. One example of these optimizers is the Rule-based Approach (RBA), which uses an intuitive set of rules to optimize queries.

Indeed, some properties allow modifying the order of the algebraic operators in order to obtain performance gains. We have, for example:

- The grouping of selections, which allows several restrictions to be carried out in a single browse of the table instead of performing one per restriction. The selection (Restriction (x = a), y = b) <=> Restriction (x = a AND y = b);
- The descent of selections and projections in an algebraic tree, making it possible to reduce the size of relations and intermediate results.
- The commutativity of restrictions and joins, which allows restrictions to be applied before joins;
- The associativity of the joins, making it possible to change the order of the joins to use more efficient algorithms in some cases, etc.

This type of optimization is easy to implement, which is why it has been considered in all commercial and academic DBMSs. The main limitation of this optimization is that it does not consider the physical parameters of the database and the deployment platform.

- ODAP optimizers appeared to overcome the shortcomings of ODCP optimizers. They take into account conceptual, logical, physical and deployment parameters. As an example of these optimizers, we have the approach based on mathematical cost models, called the Cost-Based Approach (CBA) [10, 13]. This approach consists of first calculating the cost of the different possible strategies corresponding to the execution plans of a query according to the characteristics of the files on which the relationships are established and then choosing the plan with the minimum cost. One of the actions of ODAP optimizers is to order joins (and set operations) to process the fewest tuples possible, mobilize the existing indexes as much as possible, and use the most efficient algorithms.

For this purpose, the optimization engine needs parameters from all phases of the life cycle (Figure 1):

- The conceptual layer: the length of each attribute of each table;
- The logical layer: the size (in terms of instances) of each table;
- The physical layer: the storage model used to store tables (row store, column store) and data access methods;
- Deployment layer: disk characteristics such as the size of a disk page.

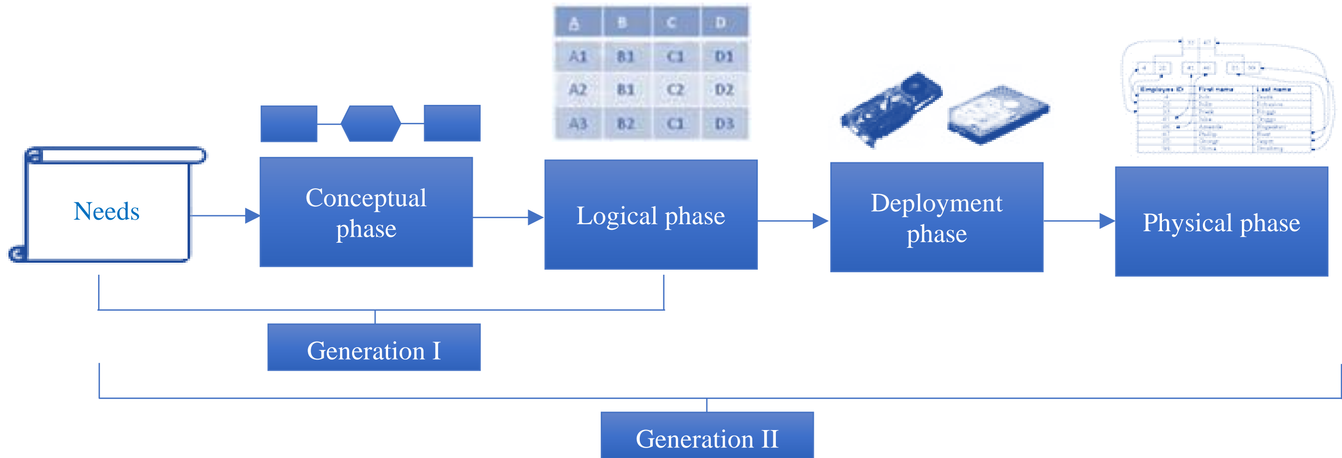

Fig. 1 DB lifecycle phases and optimization process

Thus, ODAP query optimization is sensitive to all phases of the database design lifecycle. Suppose the query language, database type, storage model, or deployment platform changes; then, the optimization process should be revisited. All optimizations are often addressed in the physical phase of the lifecycle.

Non-object-oriented databases only take one parameter from the conceptual phase: the data dictionary. Remember that the conceptual model intelligibly expresses application needs and domain knowledge for a subsequent user. Unfortunately, the logic model is exploited, resulting from normalization (in the case of relational databases) and adaptation to the support system, which is generally very different from the conceptual model. In object-oriented databases, optimizers consider the object model's characteristics in the optimization: path expressions, object groupings, pointer scans, path indexes, and user methods [24].

In the rest of this paper, methods and materials are presented at point 2. Point 3 talks about the results, and point 4 about the discussion focuses on the physical design tools most used by DBMS and their applicability in the different types of DB. Finally, the paper ends with a conclusion.

2. Materials and Methods

This article has two main objectives; the first objective is to provide a narrative review and analysis of the research studies focusing on physical design to optimize queries. The second aim is to report some problems when considering a physical design tool. The literature review is presented in a historical chronological of physical design tools. So, it starts from the indexing technique, the first optimization technique, to the more recent techniques. Redundant optimization structures and non-redundant optimization structures are considered:

- Redundant optimization structures are structures that require memory space for storage and have maintenance costs. Examples of such structures include indexes [25], materialized views [26], replication and vertical fragmentation [27].
- Non-redundant optimization structures are those that do not require storage space. These are, for example, horizontal fragmentation [28] and parallel processing without replication [27].

Main redundant optimization structures, namely indexes, materialized views and parallel processing, main non-redundant structures, horizontal fragmentation and clustering, are presented hereby.

3. Results and Discussion

Physical design tools are also called optimization structures. Some of these structures intervene very early in the lifecycle of DB; this is the case of fragmentation, parallel processing and clustering, which are considered from the conceptual phase before being created during the deployment phase. Others occur near the end of their lifecycle; this is the case of indexes and materialized views created during the exploitation phase.

3.1. Indexes

Indexing techniques are a very important option for database tuning for traditional and advanced databases. They are the oldest physical design techniques. Indexes are used to improve data access time [25]. The definition of indexes is often done while the database is running. A significant number of indexes have been proposed and supported by commercial and academic DBMSs. In the context of traditional databases, B-tree indexes, hash indexes, join indexes, bitmap indexes, etc., have been proposed [25]. Some indexing techniques have arisen in the context of data warehouses due to the nature of Online Analytical Processing (OLAP) queries. These are, for example, binary indexes, binary join indexes, star join indexes, etc. [29]. Indexes have the advantage of speeding up information searches. Indeed, an index represents the table sorted on a given field. But

indexes also have drawbacks: each time an index is created, the DBMS workload increases. Thus, data entry and maintenance operations are slowed down by the presence of indexes because they must be updated immediately. An index takes up space on the disk. Selecting a set of indexes to optimize a given query load is a difficult problem [30].

To answer this problem, the first works proposed solutions following the ODCP approach, where certain rules, such as using candidate attributes for indexing and the frequency of queries, have been proposed [31]. These rules are not sufficient to offer a successful selection. To overcome this type of selection, other works have formalized the Index Selection Problem (ISP) as an optimization problem following the ODAP approach. Since given:

- A load of queries Q, where each query has an access frequency,
- A database or datawarehouse diagram deployed on a given platform,
- An index storage space S.

The problem of index selection is to find a better configuration of index, CI, allowing to optimize Q, that is to say, to reduce the total cost of running queries while respecting the storage constraint. This problem is NP-complete [30]. Several approaches to solving this problem have been developed. They start by listing the entire candidate attributes for indexing. These attributes are chosen from those present in the WHERE, GROUP BY and ORDER BY clauses of the queries. Algorithms for selecting attributes that can be indexed have been proposed; they are guided by a mathematical cost model quantifying the quality of the solution obtained. These include genetic algorithms, simulated annealing, or full linear programming. One of these approaches is used in the What-if module of SQL Server DBMS [32]. Indexes are optimization structures that have gone through different types of databases. They have been applied to hierarchical databases and CODASYL networks to simplify the search for records. They are used and diversified in relational databases and data warehouses to facilitate access to tuples [25, 29-30]. In OODBs, in addition to classical indexes, path indexes have been proposed [19-18]. In NoSQL databases, indexes are created to guarantee a certain performance [33].

3.2. Materialized Views

A Materialized View (MV) is a named query, the result of which is stored in the database permanently. Materialized views improve query execution by precomputing the most expensive operations, such as joining and storing their results in the database. Thus, some queries only require access to materialized views and, therefore, are executed faster [26]. Materialized views are used to meet several objectives, such as improving the performance of queries or providing duplicate data (cache in proxy servers, for example). The use of MVs has three major problems, namely (i) the problem of their selection, (ii) the problem of their maintenance and (iii) the rewriting of queries based on views.

3.2.1. The Problem of their Selection

As all the views cannot be materialized for storage reasons, the selection of materialized views consists of choosing a subset of candidate views, making it possible to reduce the cost of running a load of queries. The selection of views can be made under certain constraints, generally storage space and/or a maintenance time threshold not to be exceeded. The problem of selecting materialized views can, therefore, be formalized [34-35] as given:

- A load of queries Q where each query has an access frequency;
- A DB or datawarehouse schema deployed on a given platform;
- A set of constraints C (storage space, maintenance time, etc.);

The Materialized View Selection Problem (MVSP) consists of finding a set of views, which reduces the total cost of executing the queries of Q and respects the constraints of C.

Like the index selection problem, the MVSP is NP-complete [26]. MVSP in data warehouses has been extensively studied for both the MOLAP (Multidimensional OLAP) and the ROLAP (Relational OLAP) approach. In the MOLAP approach, the data cube is considered the main structure for selecting materialized views. Each cell in the cube is considered a potential view. In the ROLAP approach, each query is represented by an algebraic tree [36]. Each node (not leaf) is considered a potential view.

3.2.2. The Problem of MV Maintenance

The database tables change and evolve at the rate of updates. However, if these changes are not carried over to the materialized views, their contents will become obsolete, and their objects will no longer represent reality. The maintenance of materialized views consists of transferring the modifications on the database tables to the level of the materialized views. This can be done using three approaches: periodic, immediate and deferred. In the periodic approach [37], views are updated continuously at specific times. In the immediate approach [38], views are updated immediately at the end of each transaction. In the latter approach, changes are propagated in a deferred fashion [39].

The maintenance of the views can be performed by recalculating these views from the database tables. However, this approach is very expensive. Good maintenance of views is performed when changes (insertions, deletions, modifications) made in the source tables can be propagated to views without recalculating their content completely. To solve this problem, three types of maintenance have been proposed: incremental, autonomous and in batch [38].

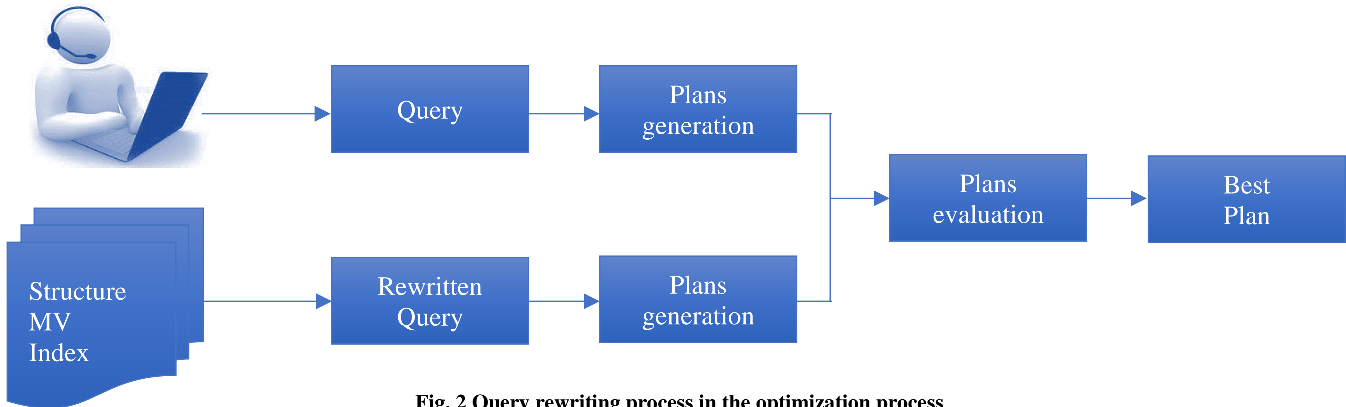

Fig. 2 Query rewriting process in the optimization process

Incremental maintenance consists of identifying the new set of tuples to be added to the view in the case of an insertion or the subset of tuples to be removed from the view in the case of a deletion without fully re-evaluating the view. Autonomous maintenance ensures that the maintenance of a view V can be calculated only from V and the changes that have occurred in the database tables on which it is defined. Batch maintenance is performed using updated transactions.

3.2.3. Rewriting Queries using Materialized Views

After selecting and creating views, all queries must be rewritten based on views. But, finding the best rewrite for a query is a difficult task [40]. The process of rewriting queries has been used as an optimization technique to reduce the cost of evaluating a query [40]. Figure 2 shows the rewrite process in the optimization process. The system evaluates the different execution plans (with MVs and without MVs) and selects the best optimal.

3.3. Horizontal Fragmentation

Horizontal fragmentation consists of partitioning a table according to its tuples to reduce the number of accesses not necessary to process queries. Two types of horizontal fragmentation exist primary fragmentation [42] and derivative fragmentation [29]. The primary horizontal fragmentation of a table is based on attributes defined on that table. Derived horizontal fragmentation is the propagation of fragmentation from one table to another table.

Horizontal fragmentation was originally proposed as a logical design technique for distributed databases in the 1980s [43]. Unlike other optimization structures like indexes and materialized views, where the selection of optimization schema is usually made when the database is operational (or created), the selection of a fragmentation schema of a database (or a datawarehouse) must be decided before its creation. This situation makes its selection more sensitive than the other structures. In addition, it can also be combined with other optimization structures such as indexes [44], materialized views [28] and parallel processing [45].

A significant amount of work on horizontal fragmentation has been developed in the context of traditional and advanced databases [43-45]. These works followed both optimization approaches (ODCP and ODAP). Early works used criteria such as affinities between selection predicates in a query payload to determine the fragmentation schema. Horizontal fragmentation is efficient if defined on attributes in the selection predicates. The affinity between two predicates is the sum of the access frequencies of queries using both predicates simultaneously. This approach is, therefore, less complex than that based on predicates. However, it only considers the frequency of access as a grouping criterion. However, to fragment a database, other parameters must be considered, such as predicate selectivity factors, the size of the table, the size of the disk page, the number of pages occupied by this table, etc. Approaches based on predicates and affinities do not provide any metrics to assess the quality of the resulting schema. To overcome these shortcomings, algorithms following the ODAP approach have been proposed in the context of object-oriented databases and data warehouses [43-45]. These algorithms include a generator of fragmentation patterns, a cost model and a module for selecting the optimal pattern. Fragmentations are used in data warehouses [43], relational databases [28] and Object-oriented DB [22].

3.4. Parallel Processing

Massively parallel processing is the use of a large number of processors or computers to perform a set of coordinated calculations in parallel (i.e. simultaneously). In the context of databases, mainly in Big Data, parallel processing is considered a technique for query optimization [48, 49]. Different approaches have been used to implement massively parallel processing, including:

- Grid Computing: in this approach, the computing power of a large number of distributed computers is used opportunistically whenever a computer is available [46] for processing a query.
- Computer Cluster: this approach uses a large number of processors located close to each other to process a query.

In such a centralized system, the speed and flexibility of the interconnection of the processors is very important.

In the process of optimizing queries in parallel processing, the optimal sequential execution plan generated at the end of the optimization phase is transformed into a parallel execution plan following the parallelization step. For the parallel execution of queries, there are two levels of parallelism: inter-query parallelism and intra-query parallelism.

- Inter-query parallelism consists of using several processors in the architecture to execute several queries at the same time;
- Intra-query parallelism consists of using several processors in the architecture to execute a given query. There are two forms of intra-query parallelism: inter-operation parallelism and intra-operation parallelism. Inter-operation parallelism is when we run multiple operations of the same query simultaneously. These operations are either independent or consecutive. Intra-operator parallelism breaks down a given operation into a set of tasks, each of which is placed on a single processor and performs the algorithm operation on the part of the data [47]. Some operators, like selection and projection, can be easily broken down into parallel tasks. Others, like the join or the decomposition, are more complex.

To facilitate the parallel processing of large data sets, MapReduce, a powerful algorithmic model using two functions (map and reduce), has been proposed [48, 49]. The queries are specified as Map and Reduce functions [48]. Tools have been developed to represent the parallel execution plan by a set of dependent MapReduce jobs [49]. A MapReduce job contains a Map phase and a Reduce phase. Each phase is instantiated by a set of parallel Map or Reduce type tasks. However, this model suffers from the problem of systematic reading and writing on a distributed file system between two jobs; this slows down query execution time.

Another model has been proposed and integrated into some tools like Hive [50] and SparkSQL [51]. Reading from the distributed file system is done only at the start of query execution. The writing is done only at the end of the execution of the query.

Relational operators are grouped into stages [52]. A set of parallel tasks instantiates each stage. A given task executes all the stadium operators on some input data. The communication between the tasks of two stages which follow one another is done either by diffusion (i.e. a tuple generated by a task of the producer stage is sent to all the tasks of the consumer stage) or by distribution (i.e. a tuple generated by a task of the producer stage is sent to a single task of the consumer stage).

3.5. Object Grouping or Clustering

Object grouping is a technique used to optimize object-oriented databases. It consists of storing on the same page objects linked by an association in order to speed up access to these objects by navigation or join [53-54]. Grouping techniques allow objects from different collections to be placed in the same group. Grouping allows an autonomous life of linked objects, which can exist without associated objects. In the case of objects without a master or objects shared by several masters, the choice of the storage page is not obvious. It is possible to define groups by selection or association predicates to capture a broad class of grouping strategies. A selection predicate makes it possible, for example, to define a group according to the selection predicate. An association predicate makes it possible, for example, to define a group for each type of object. Gruber et al. [53] proposed a data structure called a grouping graph to visualize the specification information of groups. It is a graph whose nodes represent the extensions of classes and the edges, the predicates of links used for grouping, each edge having a priority varying from 0 to 10. Gardarin et al. [54] proposed grouping techniques with priority and the model cost for object-oriented DB.

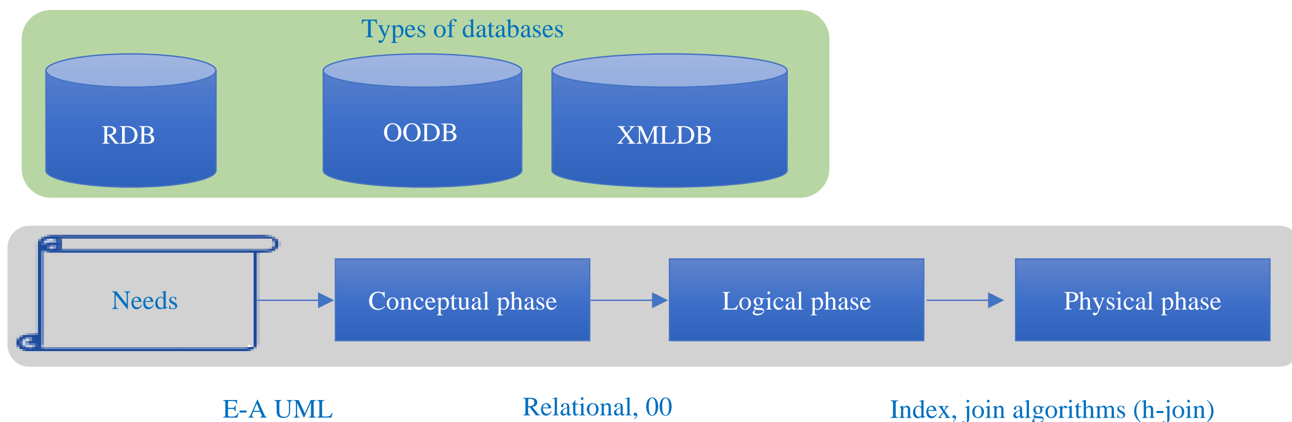

Fig. 3 Classic database lifecycle phases

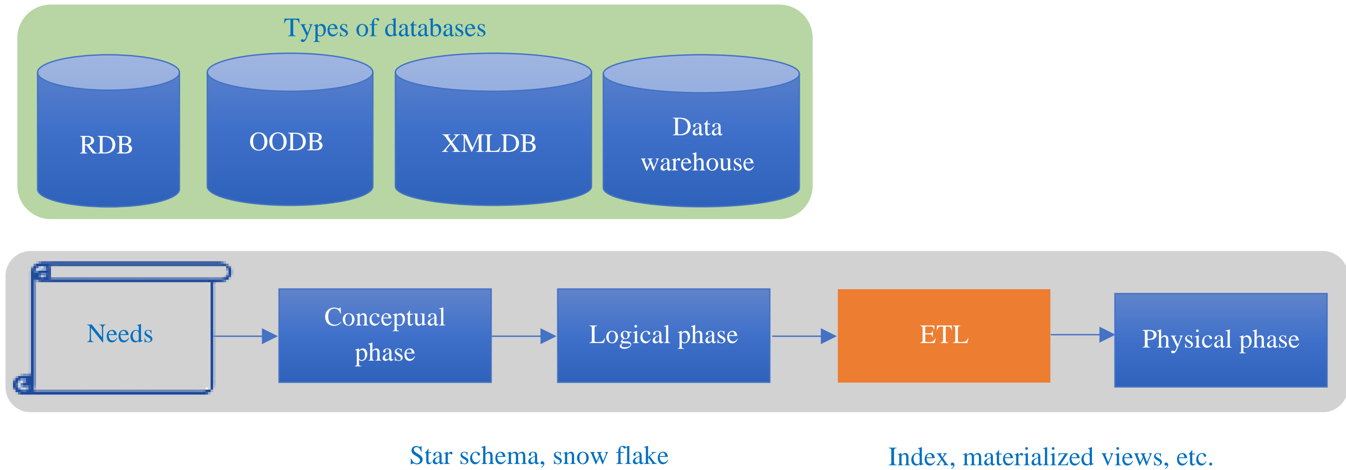

Fig. 4 Databases and data warehouse lifecycle phases

4. Discussion

Indexes turn out to be a primary physical design tool and have the privilege of being considered in all types of databases. They are used in the first databases (hierarchical and network databases), relational, object-oriented, XML databases, warehouses and new types of databases (NoSQL databases and Big Data). Materialized views are very useful in situations where they deal with expensive queries and need to be partially or fully calculated in advance [55]. They appeared with data warehouses, but their design is used beyond DBs, for example, in proxy servers and Web caching. Indexes and Materialized views can be used together to optimize data warehouses [56]. Fragmentation is a solution that consists of bringing closer to user data that he tends to query frequently. It is more interesting in a distributed environment. As for parallel processing, their use has increased with new types of databases, in particular, NoSQL databases and MapReduce type programming. They have a bright future.

Database's physical design has evolved as new database types emerge. We observe a vertical evolution of the database's physical design, resulting in the addition of new phases when a new database type appears, and a horizontal evolution of the physical design, resulting in the enrichment of each phase of the DB lifecycle by considering new tools. Figure 3 gives the phases of classical databases (BDR, BDOO and BD XML). The ETL phase was born with the advent of data warehouses, and new logical models (star schema and snowflake schema) were born. New physical design tools (materialized views) were proposed.

Fig. 4 gives an illustration. The same thing can be observed in Big Data, vertically with the apparition of multiple NoSQL database types and horizontally with the advent of the map-reduce technique. Resource reallocation based on Service Level Agreement (SLA) appears as a physical design with the database in the cloud [57].

Data administrator has to deal with these different tools to face the query optimization problem according to the used database type.

5. Conclusion

Database tuning is a very important phase to guarantee DB performance because it allows the implementation of tools to obtain query results quickly. It has received a lot of attention from the DB community. In this paper, the optimization process, which is the focal point of database tuning, is looked at. It is brought closer to the DB lifecycle and found that modern physical design techniques consider all stages of the DB lifecycle. Finally, an overview of physical design tools is presented, and their selection problems in a formal way are exposed. Physical design is not a stagnant process, i.e. it does not stop once the structures are chosen and created. It requires regular monitoring and reassessment to adapt structures to changes in data and platform parameters. The physical design has been enriched during the vertical evolution of the database with the apparition of new phases and during the horizontal evolution of DB with the development of new tools and algorithms. Data administrators must consider these tools according to the database type in the tuning work.

References

[1] Dimitri Theodoratos, and Timos Sellis, “Designing Data Warehouses,” *Data and Knowledge Engineering*, vol. 31, no. 3, pp. 279-301, 1999. [CrossRef] [Google Scholar] [Publisher Link]

[2] Mehrad Eslami et al., “Query Batching Optimization in Database Systems,” *Computers and Operations Research*, vol. 121, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[3] Surajit Chaudhuri, and Vivek Narasayya, "Self-tuning Database Systems: A Decade of Progress," *Proceedings of the 33rd International Conference on Very Large Data Bases*, pp. 3-14, 2007. [Google Scholar] [Publisher Link]

[4] Bailu Ding, Surajit Chaudhuri, and Vivek Narasayya, "Bitvector-Aware Query Optimization for Decision Support Queries," *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 2011-2026, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[5] Robert M. Pecherer, "Efficient Evaluation of Expressions in a Relational Algebra," *Association for Computing Machinery Pacific*, vol. 75, pp. 44-49, 1975. [Google Scholar] [Publisher Link]

[6] John Miles Smith, and Philip Yen-Tang Chang, "Optimizing the Performance of a Relational Algebra Database Interface," *Communications of the ACM*, vol. 18, no. 10, pp. 568-579, 1975. [CrossRef] [Google Scholar] [Publisher Link]

[7] P.A.V. Hall, "Optimization of a Single Relational Expression in a Relational Data Base Management System," *IBM Journal of Research and Development*, vol. 20, no. 3, pp. 247-257, 1976. [CrossRef] [Google Scholar] [Publisher Link]

[8] Frank P. Palermo, "A Database Search Problem," *Information Systems*, pp. 67-101, 1974. [CrossRef] [Google Scholar] [Publisher Link]

[9] Leo R. Gotlieb, "Computing Joins of Relations, *Proceedings of the 1975 ACM SIGMOD International Conference on Management of Data*, pp. 55-63, 1975. [CrossRef] [Google Scholar] [Publisher Link]

[10] S. Bing Yao, "Optimization of Query Evaluation Algorithms," *ACM Transactions on Database Systems*, vol. 4, no. 2, pp. 133-155, 1979. [CrossRef] [Google Scholar] [Publisher Link]

[11] Eugene Wong, and Karel Youssefi, "Decomposition-a Strategy for Query Processing," *ACM Transactions on Database Systems*, vol. 1, no. 3, pp. 223-241, 1976. [CrossRef] [Google Scholar] [Publisher Link]

[12] M.M. Astrahan et al., "System R: Relational Approach to Database Management," *ACM Transactions on Database Systems*, vol. 1, no. 2, pp. 97-137, 1976. [CrossRef] [Google Scholar] [Publisher Link]

[13] P. Griffiths Selinger et al., "Access Path Selection in a Relational Database Management System," *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, pp. 23-34, 1979. [CrossRef] [Google Scholar] [Publisher Link]

[14] Ashok K. Chandra, and Philip M. Merlin, "Optimal Implementation of Conjunctive Queries in Relational Data Bases," *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, pp. 77-90, 1977. [CrossRef] [Google Scholar] [Publisher Link]

[15] Alfred Aho, Y. Sagiv, and Jeffrey Ullman, "Equivalences among Relational Expressions," *Society for Industrial and Applied Mathematics Journal on Computing*, vol. 8, no. 2, pp. 218-246, 1979. [CrossRef] [Google Scholar] [Publisher Link]

[16] Yehoshua Sagiv, and Mihalis Yannakakis, "Equivalences among Relational Expressions with the Union and Difference Operators," *Journal of the Association for Computing Machinery*, vol. 27, no. 4, pp. 633-655, 1980. [CrossRef] [Google Scholar] [Publisher Link]

[17] Won Kim, Kyung-Chang Kim, and Alfred Dale, *Indexing Techniques for Object-Oriented Databases*, Object-Oriented Concepts, Databases, and Applications, pp. 371-394, 1987. [CrossRef] [Google Scholar] [Publisher Link]

[18] Alfons Kemper, and Guido Moerkotte, "Advanced Query Processing in Object Bases Using Access Support Relations," *Proceedings of the 16th International Conference on Very Large Data Bases*, pp. 290-301, 1990. [Google Scholar] [Publisher Link]

[19] E. Bertino, and W. Kim, "Indexing Techniques for Queries on Nested Objects," *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, no. 2, pp. 196-214, 1989. [CrossRef] [Google Scholar] [Publisher Link]

[20] E. Bertino, "An Indexing Technique for Object-Oriented Databases," *Proceedings Seventh International Conference on Data Engineering*, pp.160-170, 1991. [CrossRef] [Google Scholar] [Publisher Link]

[21] M. Tamer Ozsu, and Jose A. Blakeley, "Query Processing in Object-Oriented Database Systems," *Modern Database Systems*, pp. 1-19, 1995. [Google Scholar] [Publisher Link]

[22] Georges Gardarin, Jean-Robert Gruser, and Zhao-Hui Tang, "A Cost Model for Clustered Object-Oriented Databases," *Proceedings of 21st International Conference on Very Large Databases*, pp. 323-334, 1995. [Google Scholar] [Publisher Link]

[23] Rosana S.G. Lanzelotte, and Patrick Valduriez, "Extending the Search Strategy in a Query Optimizer," *Proceedings of the 17th International Conference on Very Large Data Bases*, vol. 91, pp. 363-373, 1991. [Google Scholar] [Publisher Link]

[24] E. Bertino et al., "Object-Oriented Query Languages: The Notion and the Issues," *IEEE Transactions on Knowledge and Data Engineering*, vol. 4, no. 3, pp. 223-237, 1992. [CrossRef] [Google Scholar] [Publisher Link]

[25] Himanshu Gupta et al., "Index Selection for OLAP," *Proceedings 13th International Conference on Data Engineering*, pp. 208-219, 1997. [CrossRef] [Google Scholar] [Publisher Link]

[26] Himanshu Gupta, "Selection of views to Materialize in a Data Warehouse," *International Conference on Database Theory*, pp. 98-112, 1997. [CrossRef] [Google Scholar] [Publisher Link]

[27] Alexandre A.B. Lima et al., "Parallel OLAP Query Processing in Database Clusters with Data Replication," *Distributed and Parallel Databases*, vol. 25, pp. 97-123, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[28] Sanjay Agrawal, Vivek Narasayya, and Beverly Yang, "Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design," *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pp. 359-370, 2004. [CrossRef] [Google Scholar] [Publisher Link]

[29] Kamel Boukhalfa, "*From Physical Design to Data Warehouse Administration and Tuning Tools*," Isae-Ensma Ecole Nationale Supérieure de Mécanique et d'Aérotechique - Poitiers, 2009. [Google Scholar] [Publisher Link]

[30] Douglas Comer, "The Difficulty of Optimum Index Selection," *ACM Transactions on Database Systems*, vol. 3, no. 4, pp. 440-445, 1978. [CrossRef] [Google Scholar] [Publisher Link]

[31] Nicolas Pasquier et al., "Discovering Frequent Closed Itemsets for Association Rules," *International Conference on Database Theory*, pp. 398-416, 1999. [CrossRef] [Google Scholar] [Publisher Link]

[32] Surajit Chaudhuri, and Vivek Narasayya, "Auto Admin "what-if" Index Analysis Utility," *ACM SIGMOD Record*, vol. 27, no. 2, pp. 367-378, 1998. [CrossRef] [Google Scholar] [Publisher Link]

[33] Rupali Chopade, and Vinod Pachghare, "MongoDB Indexing for Performance Improvement," *ICT Systems and Sustainability*, pp. 529-539, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[34] Himanshu Gupta, *Selection and Maintenance of Views in a Data Warehouse*, Stanford University, pp. 1-114, 1999. [Google Scholar] [Publisher Link]

[35] Jeffrey D. Ullman, "Efficient Implementation of Data Cubes Via Materialized Views," *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 386-388, 1996. [Google Scholar] [Publisher Link]

[36] Philip A. Bernstein, and Dah-Ming W. Chiu, "Using Semi-Joins to Solve Relational Queries," *Journal of the Association for Computing Machinery*, vol. 28, no. 1, pp. 25-40, 1981. [CrossRef] [Google Scholar] [Publisher Link]

[37] Michel E. Adiba, and Bruce G. Lindsay, "Database Snapshots," *Proceedings of the Sixth International Conference on Very Large Data Bases*, vol. 6, pp. 86-91, 1980. [Google Scholar] [Publisher Link]

[38] Jose A. Blakeley, Per-Ake Larson, and Frank Wm Tompa, "Efficiently Updating Materialized Views," *ACM SIGMOD Record*, vol. 15, no. 2, pp. 61-71, 1986. [CrossRef] [Google Scholar] [Publisher Link]

[39] A. Segev, and J. Park, "Maintaining Materialized views in Distributed Databases," *Proceedings Fifth International Conference on Data Engineering*, pp. 262-270, 1989. [CrossRef] [Google Scholar] [Publisher Link]

[40] Divesh Srivastava et al., "Answering Queries with Aggregation Using Views," *Proceedings of the 22th International Conference on Very Large Data Bases*, pp. 318-329, 1996. [Google Scholar] [Publisher Link]

[41] Pavan Edara, and Mosha Pasumansky, "Big Metadata: When Metadata is Big Data," *Proceedings of the VLDB Endowment*, vol. 14, no. 12, pp. 3083-3095, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[42] Sharma Chakravarthy et al., "An Objective Function for Vertically Partitioning Relations in Distributed Databases and its Analysis," *Distributed and Parallel Databases*, vol. 2, pp. 183-207, 1994. [CrossRef] [Google Scholar] [Publisher Link]

[43] Stefano Ceri, Mauro Negri, and G. Pelagatti, "Horizontal Data Partitioning in Database Design," *Proceedings of the 1982 ACM SIGMOD International Conference on Management of Data*, pp. 128-136, 1982. [CrossRef] [Google Scholar] [Publisher Link]

[44] Pankaj Gupta, and Prakashkumar Patel, "Demystifying Databases: Exploring their Use Cases," *SSRG International Journal of Computer Science and Engineering*, vol. 10, no. 6, pp. 43-53, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[45] Thomas Stöhr, Holger Märtens, and Erhard Rahm, "Multi-Dimensional Database Allocation for Parallel Data Warehouses," *Proceedings of the 26th International Conference on Very Large Data Bases*, pp. 273-284, 2000. [Google Scholar] [Publisher Link]

[46] Radu Prodan, and Thomas Fahringer, *Grid Computing: Experiment Management*, *Tool Integration, and Scientific Workflows*, Springer, pp. 1-317, 2007. [Google Scholar] [Publisher Link]

[47] Bonneau Sophie, and Hameurlain Abdelkader, "*Placement of SQL Query(s) on a Parallel Distributed Memory Architecture: From Static to Dynamic*," PhD Thesis, University of Toulouse, pp. 1-213, 1999. [Google Scholar] [Publisher Link]

[48] Jeffrey Dean, and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, 2008. [CrossRef] [Google Scholar] [Publisher Link]

[49] Tom White, *Hadoop: The Definitive Guide*, O'Reilly, pp. 1-657, 2012. [Google Scholar] [Publisher Link]

[50] Bikas Saha et al., "Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications," *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 1357-1369, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[51] Michael Armbrust et al., "Spark SQL: Relational Data Processing in Spark," *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 1383-1394, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[52] Avrilia Floratou, Umar Farooq Minhas, and Fatma Özcan, "SQL-on-Hadoop: Full Circle Back to Shared-Nothing Database Architectures," *Proceedings of the Very Large Data Bases Endowment*, vol. 7, no. 12, pp. 1295-1306, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[53] Olivier Gruber, and Laurent Amsaleg, "Object Grouping in EOS," *Unite De Recherche Inria-Rocquencourtpp*, pp. 1-20, 1992. [Google Scholar] [Publisher Link]

[54] Georges Gardarin, Jean-Robert Gruser, and Zhao-Hui Tang, "Cost-Based Selection of Path Expression Processing Algorithms in Object-Oriented Databases," *Very Large Data Base*, pp. 1-26, 1996. [Google Scholar] [Publisher Link]

[55] Mitesh Athwani, "A Novel Approach to Version XML Data Warehouse," *SSRG International Journal of Computer Science and Engineering*, vol. 8, no. 9, pp. 5-11, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[56] Abbassi Kamel, and Tahar Ezzedine, "Dynamic Selection of Indexes and Views Materialize with Algorithm Knapsack," *2019 International Conference on Internet of Things*, *Embedded Systems and Communications (IINTEC)*, pp. 214-219, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[57] Mohamed Mehdi Kandi, Shaoyi Yin, and Abdelkader Hameurlain, "SLA-Driven Resource Re-Allocation for SQL-Like Queries in the Cloud," *Knowledge and Information Systems*, vol. 62, pp. 4653-4680, 2020. [CrossRef] [Google Scholar] [Publisher Link]