Computer Science and Information Systems 19(2):735–762 https://doi.org/10.2298/CSIS210606003M

ProRes: Proactive Re-Selection of Materialized Views

Mustapha Chaba Mouna[1], Ladjel Bellatreche[2], and Narhimene Boustia[1]

[1] LRDSI Laboratory, Faculty of Science, University Blida 1
Blida, Algeria {mustapha.medea, nboustia}@gmail.com
[2] LIAS/ISAE-ENSMA, Poitiers, France
bellatreche@ensma.fr

Abstract. Materialized View Selection is one of the most studied problems in the database field, covering SQL and NoSQL technologies as well as different deployment infrastructures (centralized, parallel, cloud). This problem has become more complex with the arrival of data warehouses, being coupled with the physical design phase that aims at optimizing query performance. Selecting the best set of materialized views to optimize query performance is a challenging task. Given their importance and the complexity of their selection, several research efforts both from academia and industry have been conducted. Results are promising – some solutions are being implemented by commercial and open-source DBMSs –, but they do not factor in the following properties of nowadays analytical queries: **(i)** large-scale queries, **(ii)** their dynamicity, and **(iii)** their high interaction. Studies to date fail to consider that complete set of properties. Considering the three properties simultaneously is crucial regarding today's analytical requirements, which involve dynamic and interactive queries. In this paper, we first present a concise state of the art of the materialized view selection problem (VSP) by analyzing its ecosystem. Secondly, we propose a proactive re-selection approach that considers the three properties concurrently. It features two main phases: offline and online. In the offline phase, we manage a set of the first queries based on a given threshold δ by selecting materialized views through a hypergraph structure. The second phase manages the addition of new queries by scheduling them, updates the structure of the hypergraph, and selects new views by eliminating the least beneficial ones. Finally, extensive experiments are conducted using the Star Schema Benchmark data set to evaluate the effectiveness and efficiency of our approach.

Keywords: Materialized Views, Hypergraphs, Query Sharing, large-scale of queries.

1. Introduction

Selection of materialized views is a challenging task for designing advanced database applications such as Analytical Databases [24], Autonomous Databases [2], Semantic Databases [22], Cloud Databases [10] and NoSQL Databases [58]. The idea of using materialized views to satisfy the quality-of-service of databases does not date from today, but since forty years [35]. Their importance has been amplified since data warehouse physical design has become more sophisticated to cope with complex decision support queries [14,4]. Selecting a set of materialized views that would satisfy functional and non-functional requirements is complex [24]. This gave rise to the materialized view selection problem (VSP). Due to the importance and complexity of this selection, several research efforts from academia and industry have been conducted. Certainly, the results obtained

by these efforts have been implemented in commercial and open-source DBMSs such as Data Tuning Advisor for SQL Server [1], Design Advisor for DB2 [64], SQL Access Advisor for Oracle, and Parinda for PostgreSQL [39].

By examining the major solutions of VSP, we figure out that they usually consider static workload of queries . In other terms, the potential views are quantitatively evaluated and then greedily pre-materialized prior to executing the query workloads [49]. Formally, the VSP is defined in the literature as follows: given a workload of queries $Q = \{Q_1, Q_2, \ldots, Q_n\}$ and a set of resource constraints C (e.g., storage cost and maintenance cost). The VSP consists in selecting a set of materialized views $MV = \{V_1, V_2, \ldots, V_m\}$ that satisfies some of the non-functional requirements such as minimizing query performance, saving energy consumption, etc. and respects C.

This situation is inadequate with the nowadays requirements of analytical applications, where *high number*, *dynamic*, and *high interacted queries* are *simultaneously considered*.

Due to the importance of the above three properties of nowadays workloads containing a high number of dynamic, and highly interacted queries, their clarification is necessary. With regard to the high number of queries, let us consider the following real examples covering analytical and semantic databases: **(i)** the Periscope application manages twenty-something million queries per day[3]; **(ii)** the snowflake platform deals with more than 300 million queries per day from its customer base[4], and **(iii)** the obtained results of a recent paper published in VLDB'2020, dealing with the problem of selecting materialized views in Oracle DBMS, are obtained based on 650 queries running on a star schema [2], and **(iv)** the SPARQL query logs executed at scholarly data of DBpedia contains 43 284 queries [34].

The query operation sharing is a guiding characteristic and at the same time impacted by the two other properties. Sharing computation among multiple concurrent queries was first studied by Sellis in 1988 in the context of multi-query optimization (MQO). Recently, this principle has been reproduced in the context of Cloud Databases under the name "*Pay One, Get Hundreds for Free*" [41]. The identification of common subexpressions of queries is the key issue for the performance of multi-query processing, *even for a small set of queries*, which was the natural hypothesis of existing studies. Historically, the Problem of Multi-Query Optimization (PMQO) has been largely studied in the 80's [13], [56] in the context of relational databases [50]. This problem has been revisited in all database generations without any exception since it aims at optimizing the global performance of queries collectively instead of individually. The PMQO has been combined with several important database problems such as caching [46], materialized view selection [40], reusing [23], indexing [28], data partitioning [5], optimizing exploratory queries [32]. In the context of relational data warehouses, the PMQO has been amplified since OLAP workloads are a windfall of query sharing, where typical OLAP and reporting workloads are overlapping.

Regarding the dynamic aspect of OLAP queries, the diversity of data analysis and the changing business directives make the query workload more dynamic [53]. This dynamicity has contributed to increasing the research studies on self-tuning and autonomous databases [2,11,47]. Several reactive approaches such as *DynaMat* [33], *WATCHMAN*

[3] https://thenewstack.io/how-periscope-uses-kubernetes-to-power-data-science-services/
[4] https://diginomica.com/snowflake-ceo-insists-his-company-can-take-heat

[54], and Materialized Query Table (MQT) advisor [49] for dynamic materialized view selection have been proposed. The main characteristic of these approaches is that they react to transient usage, rather than purely relying on a historical workload [48]. These solutions have to solve multiple problems [37]: what views to materialize [49] for serving current and future queries, when to evict views (LRU cache) [49].

This motivates us to propose in this paper, a Proactive Re-selection of materialized views (called ProRes) that integrates concurrently our three properties (Figure 1). Since our queries arrive dynamically, based on a threshold, the first δ incoming queries are routed to the offline phase that selects the most beneficial common subexpressions for materialization purposes. The other coming queries are managed by the online phase that exploits the selected views of the offline one. *ProRes* integrates three fundamental aspects: **(i)** the usage of dynamic hypergraph structure that captures the interaction among queries, **(ii)** bounding intervals are used to update the current set of materialized views based on their benefits, and **(iii)** query scheduling if necessary. Contrary to most traditional studies which assume that the queries are already pre-ordered, our queries can be scheduled if their order reduces the query performance.

The paper is organized as follows: Section 2 overviews the most important studies related to our two studied problems: PMQO and VSP. Section 3 presents the basics and definitions related to hypergraphs and the processes for passing from queries to hypergraphs. Section 4 describes our ProPres approach, where all its components are detailed. Section 5 presents our intensive experiments and a real validation in commercial DBMS. Section 6 concludes the paper and discusses future work.

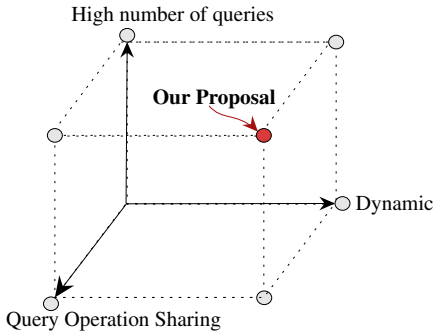


Fig. 1. Position of our Proposal according to the state-of-the-art

2. Related Work

This section discusses the most important studies dealing with PMQO and VSP either in isolation or jointly.

MQO is one of the main topics studied by the database community [52]. A specific chapter on MQO has been reserved in the encyclopedia of Database Systems edited by Ling Liu and Tamer Özsu [38]. Before connecting MQO to VSP, it would be wise to provide its separate generic formalization. For a given workload of queries to be optimized, where each query has a set of individual plans, the PMQO aims at finding the

best merging of query plans where the global query processing cost is minimized. The PMQO has proven as NP-hard in [59] where its search space formulation was given. An A* algorithm with bounding functions and intelligent state expansion, based on query order, to eliminate states of little promise rapidly has been proposed and guarantees an optimal solution, for a small set of queries [56]. Several variations of Sellis's algorithms have been proposed [59] to prune the search space of the PMQO. Genetic, simulated annealing and Depth-first Branch-and-Bound algorithms have also been experimentally analyzed to handle larger MQO problems that cannot be solved using A* in a reasonable time [16] [57]. Other algorithms are based on game theory [3]. [52] address the problem of extending top-down cost-based query optimizers to support multi-query optimization, and present greedy heuristics, as well as implementation optimizations. Their techniques were shown to be practical and to give good results. The adaptation of these findings has been reproduced in several generations of databases through their query languages: object-oriented databases (OQL) [63], semantic databases (SPARQL) [36], XML databases (XPATH) [20], distributed databases [32] [42], stream databases (CQL) [18], graph databases (Sparql 1.1) [19], data mining [44] and SQL-on-Hadoop systems [15].

MQO and physical design are the most active research topics in the field of databases and information systems. These two problems interact with the use of the query interaction which has a great impact on logical and physical optimizations. Despite their strong dependency, MQO and the instances of physical design have been tackled separately without really taking into account their interaction. The PMQO has usually been studied for a static set of queries, where several variations of the A* algorithm have been proposed for finding optimal solutions to the moderately sized (up to ten queries) [57]. Other studies advocated the fact that rather than trying to obtain an optimal solution, finding near-optimal solutions in less time is suitable. In the last years, there has been a growing interest in solving physical design problems especially in selecting materialized views, which are considered as one of the most interesting techniques to optimize OLAP queries. The VSP has been addressed in static and dynamic contexts without really considering their interaction with the MQO problem. To the best of our knowledge, The work proposed by [60] is the pioneer in the DW context that has highlighted this dependency. The main drawback of this work concerns the scalability of their algorithms in constructing the MVPP. To fill this gap, hypergraphs have been proposed in [8] to capture the query interaction among very large sets of static queries, then to study their contributions for selecting an appropriate set of materialized views. Promising results have been obtained showing the great benefit of hypergraphs to deal jointly with PMQO and the VSP [8] [51].

In physical design, Materialized views are one of the most important techniques to optimize analytical queries and are strongly dependent to query operation sharing. The materialized views selection problem is an NP-hard problem [24]. A large panoply of algorithms has been proposed to deal with this problem. We are not overviewing them, since several surveys exist. we suggest to readers the reference [51] which provides a nice classification of the existing algorithms. For complete classification of these algorithms, we recommend the readers to refer to the survey paper of [40] which divides algorithms into the following categories: deterministic algorithms, randomized algorithms, evolutionary algorithms, and hybrid algorithms. From the scope of our paper, these existing studies consider a small set of static queries, which contradicts the ad-hoc nature of analytical queries. Dynamat system [33] is one of the most popular systems that studied VSP in a

dynamic context. It monitors permanently the incoming queries and uses a pool to store the best set of materialized views based on a goodness metric and subject to space and update time constraints.

By analyzing the VSP studies, we realize that they are not connected to PMQO. The work proposed in [60] is the pioneer in the context of DW that showed the strong dependency among PMQO and VSP. It aims at constructing a unified query plan for a given set of queries. At first, they select the individual join plan for each query. Afterward, these plans are merged in a unified query plan called MVPP represented by a directed acyclic graph and dedicated to the process of selecting materialized views. The main limitation of this work is related to the scalability of their algorithms in constructing the MVPP. To counter this problem, hypergraphs have been proposed in [7] for coupling PMQO and VSP by considering the high number of queries. The authors have used the hypergraph structure for the identification of common subexpressions among a very large set of queries and to study their contributions for selecting an appropriate set of materialized views. In fact, Considering a huge number of queries produces a massive number of views to materialize which violates the storage space constraint. Indeed, it is impossible to materialize all candidate views under such a situation. To handle this problem, the authors [7] put a strategy that aims to maximize the benefit of using the materialized views before their dropping. To do so, they proposed a new query scheduling policy that allows finding the best query order that produces the highest benefit of the materialized views set.

An in-depth analysis of the major studies dealing with PMQO and VSP allows identifying several common points. The main shared point among PMQO and VSP is the use of graphs theory and their data structure either for pruning their large-scale search spaces or to identify the common sub-expressions among queries. Three main graph data structures have been proposed to prune the search space of VSP problem: AND/OR viewgraph [24] [43] , data cube lattice [25] [61] [29] and MVPP [60]. For PMQO, query graphs have been used to represent a set of queries [13] by merging the individual query trees in a single unified query plan (UQP). The UQP spans four levels of nodes: selection, join, projection and aggregation.

Another point shared by VSP and PMQO is the use of query scheduling policies to improve their algorithms. To illustrate this point, several good examples can be given. For instance, the work of [16], where the query scheduling has played a crucial role to improve the MQO algorithm. Also, several works have studied VSP under query scheduling constraints. In [49], a dynamic formalization of VSP is given by considering query scheduling policy based on a genetic algorithm. The two main limitations of Phan's algorithm [49] are (i) their dependence on DB2 advisor and (ii) their greedy genetic algorithm for reordering queries which are not suitable when scaling. To overcome these limitations, a scalable approach called *SLEMAS* is proposed in [7]. The scalability of this approach is ensured by the means of hypergraphs structure which is used to identify the query operation sharing among very large sets of queries. Afterward, the most shared operations are considered as candidates for materialization. The materialized views in SLEMAS are dynamically selected by considering scheduling constraints for a priori known workload of queries. The main drawback of this approach is their assumption of static sets of queries, which contradicts the dynamic nature of analytical queries.

Recently, PMQO and VSP are studied and revisited under a new angle by considering simultaneously the 3-characteristics of today's analytical queries [45]. To do so, dynamic

hypergraphs have been used to capture the query operation sharing, and dynamically selecting the appropriate set of materialized views without any consideration of the query scheduling constraint. Intensive experiments are conducted by this work [45] to compare the efficiency of their proposal against the major state-of-art. The obtained results showed the great impact of the query scheduling techniques in maximizing the benefit of materialized views and optimizing the performance of incoming workloads. Table 1 summarizes our discussion by showing the interest of the main studies dealing with PMQO and VSP in an isolated way or jointly.

Table 1. Classification of existing works on PMQO and VSP

Problems	**Work**	**Used Properties**	**Data Structure**	**Query Scheduling**	**Drawbacks**
MQO	[56]	Query Sharing	Query Graph	No	Scalability
	[16]	Query Sharing	Query Graph	Yes	Scalability
VSP	[33]	Dynamic	No	No	Scalability
	[49]	No one	No	Yes	Scalability and DB2 Dependence
MQO and VSP	[60]	Query Sharing	MVPP	No	Scalability
	[7]	High number of queries & Sharing	Hypergraph	Yes	Static
	[45]	three-properties	Hypergraph	No	Pre-ordered Queries

3. Background

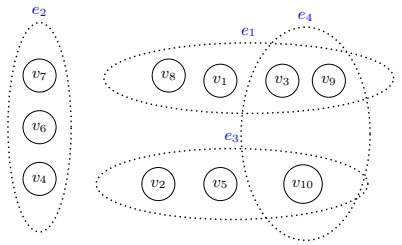
In this section, we present some fundamental notions and definitions related to hypergraphs and their ability to manage the three properties of analytical queries.

Hypergraphs are powerful tools for representing complex and non-pairwise relationships. They contributed in several domains in capturing the interaction between studied objects such as data mining, text/image retrieval, bio-informatics, social mining, and machine learning [27]. In the database fields, Hypergraphs have been used at logical and physical phases (e.g., the detection of functional dependencies [21], data partitioning for optimizing OLTP workloads [17] and materialized views selection for optimizing large-scale OLAP workloads [8]).

Definition 1. *A hypergraph $H = (V, E)$, is defined as a set of vertices V (nodes) and a set of hyper-edges E, where every hyper-edge connects a non-empty subset of nodes [9]. Note that when $|e_i| = 2$ $(\forall i = 1...m)$, the hypergraph is a standard graph.*

Definition 2. *The degree of a vertex $v_i \in V$, denoted by $d(v_i)$ represents the number of distinct hyper-edges in E that connect v_i.*

The incidence matrix of a hypergraph allows counting the connection between hyper-edges and vertices, where rows and columns represent respectively vertices and hyper-edges. The $(i, j)th$ value in the matrix, denoted by $IMij$, is equal to 1 if vertex v_i is



(a) Hypergraph H_1

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & e_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

(b) The Incidence Matrix of H_1

Fig. 2. Example of Hypergraph

connected in the hyperedge e_j, and 0 otherwise.

$$IMij = \begin{cases} 1, & \text{if jth hyperedge contains the ith vertex} \\ 0, & \text{otherwise.} \end{cases}$$

3.1. Representation of a Query by a Hypergraph

For giving a realistic hypothesis, we consider in our study the SPJ (Select-Project-Join) class of queries, which represent the most common queries studied in database theory. A particular focus on selections and joins known as costly operations.

Definition 3. *A query tree is a tree data structure representing a relational algebra expression. The tables of the query are represented as leaf nodes. The relational algebra operations are represented as the internal nodes. The root represents the query as a whole.*

In our study, the query plan of a given query is obtained by left-deep tree [26], where all selections are pushed down as far down through its query graph (tree).

In the following, we show how a query tree is transformed into a hypergraph.

Example 1. To illustrate how to represent an OLAP query by a hypergraph, let assume the following query Q defined on the star schema benchmark (SSB)[5] that contains a fact table *Lineorder* and four dimension tables *Customer*, *Supplier*, *Part*, and *Dates*.

```
select  d_year, s_nation, p_category,
sum(lo_revenue - lo_supplycost) as profit
from     DATES, CUSTOMER, SUPPLIER, PART, lineorder
where  lo_custkey = c_custkey (J2)
and lo_suppkey = s_suppkey    (J1)
and lo_PARTkey = p_PARTkey    (J4)
and lo_orderdate = d_datekey  (J3)
and c_region = 'EUROPE'
and s_region = 'EUROPE'
and d_year = 1993
and p_mfgr = 'MFGR#2'
group by  d_year, s_nation, p_category
order by  d_year, s_nation, p_category;
```

The query tree of the query Q is given in Fig.3, where four joins and selections are well represented. This tree can be easily transformed to a hypergraph with four vertices representing the four joins $\{J_1, J_2, J_3, J_4\}$ and one hyperedge e corresponding to our query Q. Note that each join node is defined by its join predicate and its associated selections.

[5] http://www.cs.umb.edu/~poneil/StarSchemaB.pdf

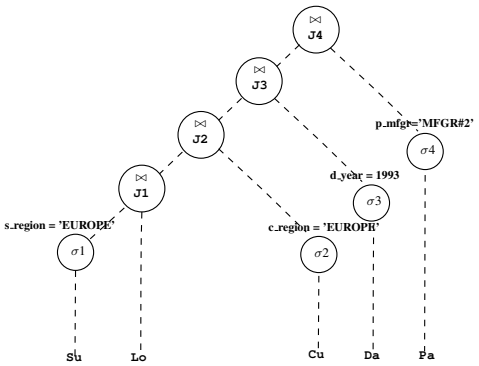
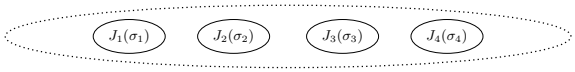


Fig. 3. The Tree of the Query Q



(a) Hypergraph of the Query Q

	e
J_1	1
J_2	1
J_3	1
J_4	1

(b) Its Incidence Matrix

Fig. 4. The Representation of the Query Q by a Hypergraph

From the query hypergraph, the algebraic tree can also be generated if the type of join processing tree (e.g., left deep, right deep, and bush) and the join order are a priori known. It should be noticed that the join order has a crucial role in optimizing star join queries involving dimension tables and a fact table. In our study, the fact table of an analytical query is always joined with dimension tables following their size (from small to large). Recent machine and deep learning-driven techniques for tackling the join order problem can be easily incorporated in our approach [62].

3.2. Hypergraph for Capturing the Interaction of Queries

In this section, we show how hypergraphs can easily capture the query sharing. For a given workload of queries W, we define a global hypergraph GH with a set of vertices GH_V and a set of hyperedges GH_E. Each vertex $v_i \in GH_V$ corresponds a join node J_i, whereas each hyperedge $e_j \in GE_E$ corresponds to a query $Q_j \in W$. The set of vertices of an hyperedge e_j represents the set of joins that participates in the processing of the query Q_j.

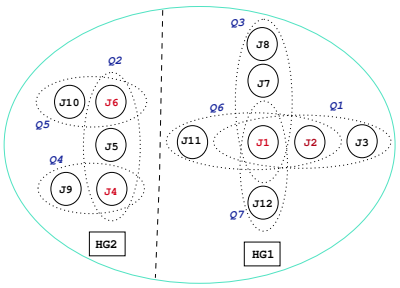
Definition 4. *A pivot node of a hypergraph is the first join shared by all queries.*

Example 2. To illustrate the construction of the global hypergraph for a given query workload, let us consider 7 OLAP queries ($\{Q_1, Q_2, Q_3, \ldots, Q_7\}$) defined in the appendix and generated randomly using the star schema benchmark query generator. Fig. 5 shows the obtained global hypergraph and its incidence matrix. The incidence matrix can easily give hints on the most shared joins by adding a column representing the usage frequency of

each join operation J_i ($Freq_i$):

$$\sum_{j=1}^{n} IM_{ij}. \tag{1}$$

Since the selection operations are performed before joins, our global hypergraph may contain several disjoint components. This is because selections reduce query sharing. In our example, the hypergraph contains two components including respectively $GH_1 : (Q_1, Q_3, Q_6, Q_7)$ and $GH_2 : (Q_2, Q_4, Q_5)$. We observe that all queries of the first component GH_1 share the same join operation identified by node J_1. As shown in the hypergraph and its incidence matrix, the four nodes identified by J_1, J_2, J_4 and J_6 are the most shared join operations and they will be considered as candidates for materialization.



(a) The Global Hypergraph

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	FRQ
j_1	1	0	1	0	0	1	1	4
j_2	1	0	0	0	0	1	0	2
j_3	1	0	0	0	0	0	0	1
j_4	0	1	0	1	0	0	0	2
j_5	0	1	0	0	0	0	0	1
j_6	0	1	0	0	1	0	0	2
j_7	0	0	1	0	0	0	0	1
j_8	0	0	1	0	0	0	0	1
j_9	0	0	0	1	0	0	0	1
j_{10}	0	0	0	0	1	0	0	1
j_{11}	0	0	0	0	0	1	0	1
j_{12}	0	0	0	0	0	0	1	1

(b) The Incidence Matrix

Fig. 5. The Global Hypergraph of the 7 Queries and its Incidence Matrix

3.3. Hypergraph for Managing High Number of Queries

The hypergraphs theory has a long history in solving many large-scale problems thanks to their ability in modeling any relationships, and their efficiency in dividing large search space of difficult problems into several sub search spaces through their fast partitioning tools. The hypergraphs have shown efficiency in managing several applications dealing with a huge amount of data and transactions. Several real examples can be given: As in VLSI design, in which hundreds of millions of gates are needed to design logical circuits through the hypergraph. Also, in social network applications, hypergraphs have been widely used to capture the interaction and behaviors of users [65].

Hypergraph partitioning is commonly used in dividing the search space of combinatorial large-scale problems into several sub-search spaces, which reducing prominently the complexity of the studied problems. This feature is ensured through the advanced tools of partitioning, which ensure scalability. Hypergraph partitioning problem consists of dividing the vertex set of the hypergraph H into a fixed number of k disjoint partitions of bounded size $\Pi = \{P_1, P_2, \ldots, P_k\}$, while minimizing a given objective function [55].

hMeTis [31], and PaToH [12] are two examples of hypergraph partitioning techniques initially developed in the VLSI domain.

Figure 6 shows an example of partitioning of the hypergraph given in Figure 2a into three partitions.

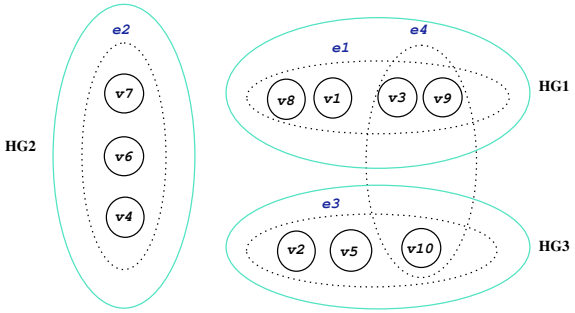


Fig. 6. An Example of Hypergraph Partitioning using hMetis

4. A Proactive Re-Selection of Materialized Views

We have all ingredients, to present our proactive re-selection of materialized views that considers our three properties. Before detailing our proposal, let us formalize our problem: given: (i) a set of queries known in advance, (ii) a set of ad-hoc queries that arrives dynamically, and (iii) a storage constraint. Our problem consists in selecting a set of materialized views speeding up the performance of all queries and satisfying the storage space constraint.

Our approach to deal with this problem is composed of two phases: offline phase that manages a set of the first δ queries Q_{first} based on the defined threshold δ and online phase that deals with the arrival of the ad-hoc queries. The global architecture of our proposal is described in Fig.7.

4.1. The Offline Phase

The Offline Phase is relies on the following main modules.

1. **Query parser**: allows parsing the set of queries Q_{first} in order to identify their logical operations (nodes) (Selection-Projection-Join).
2. **Hypergraph Construction:** once all queries of our first set of queries Q_{first} are parsed, we use the same rules described in Example 2 to construct our initial global hypergraph using two main primitives : add-node () , add-hyperedge().
3. **Hypergraph Partitioning:** Since our goal is to select joins that have a high sharing that are candidates for materialized. Therefore, our hypergraph has to be partitioned into groups of queries according to their interaction, where the query interaction is maximal inside each component and minimal among components. To ensure scalability, we adapt hMeTiS algorithm [30].
   Contrary to the original codes of hMeTiS, where the number of partitions is known in advance, in our case, the number of components to construct is unknown.
   To partition our hypergraph, we adapt an existing algorithm derived from graph theory to aggregate the join nodes into small connected components. The partition process is applied on initial hypergraph $GH(V, E)$ and the result of hypergraph partitioning is k sub-hypergraphs, where each one is an hypergraph: $GH_i(V_i, E_i)$, where $|E_i| \leq M$ $(1 \leq i \leq k)$. Our partitioning algorithm follows the same heuristic detailed in [6] that includes the following steps :

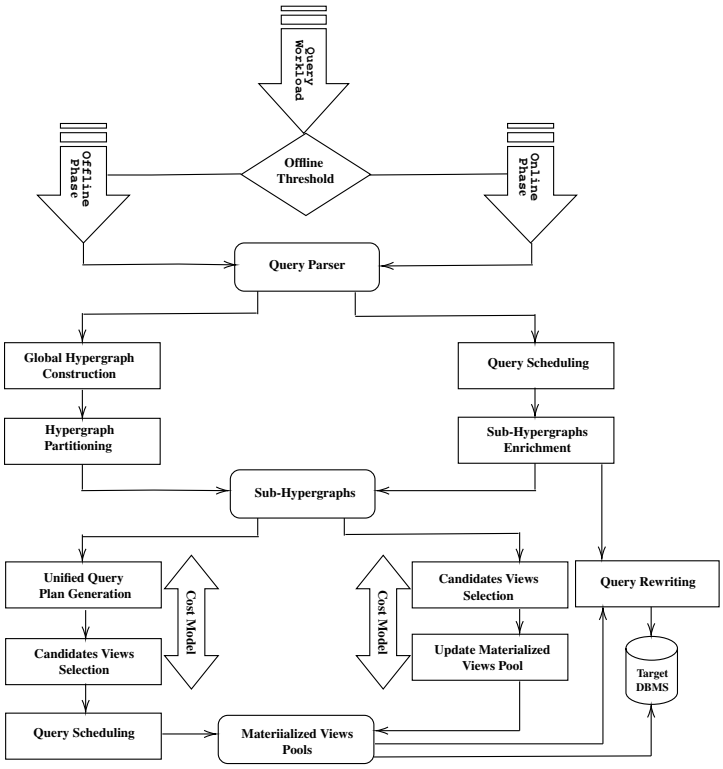


Fig. 7. The Global Architecture of our Approach

(a) Firstly, we adapt *the code of the multilevel hypergraph partitioning* Hmetis to split our hypergraph into k partitions such that the number of hyperedges cut is minimal. In our context, the exact number of partitions to construct is unknown. In the same time, we want to get all possible disjoint partitions (connected components). To do so, we adapt the original algorithm to our problem by bi-partitioning until no partition can be repartitioned without cutting hyperedges. More precisely, the algorithm behaves as follows: (i) the set of vertices will be divided if and only if the number of hyperedges cutting is null. (ii) Each bisection result of the hypergraph partitioning will be divided in the same way until no more divisible hypergraph is found.

(b) Secondly, we use Hmetis to partition each sub-hypergraph $GH_i(V_i, E_i)$, such as $|E_i| \leq M$. The sub-hypergraph $GH_i(V_i, E_i)$ is then partitioned into k' partitions such as $k' = (|E_i|/M) + 1$.

Table 2 summarizes the mapping between the *graph* vision and the *query* vision.

4. **Unified query plan Generation:** This step aims at generating the unified query plan (UQP) for each connected component by transforming each sub-hypergraph into an oriented graph. This generation is driven by cost models that allow ordering nodes. This step is necessary to order the join nodes in each component. Adding an arc to the oriented graph corresponds to putting an order between two join nodes. The transformation process has three main stages:

Table 2. Analogy Graph – Query.

Vision hypergraph	Vision of query
Set of vetices	Set of join nodes
Hyperedge	Query
sub-hypergraph	connect component
Oriented graph	Processing Plans

(a) choose the pivot node, the pivot node corresponds to the node which has the best possible benefit from reusing the intermediate results. The benefit of each node n_i is calculated using equation 2:

$$benefit(n_i) = (nbr\_use - 1) \times process\_cost(n_i) - constr(n_i) \quad (2)$$

where nbr_use, $process\_cost(n_i)$, and $constr(n_i)$ represent respectively the number of queries that use the join node ni, the processing cost of n_i, and the construction cost of n_i. In our study, we assume that the hash join is used to process join operations. The cost of a join involving two tables T_i and T_j is given by the following formula: $3 \times (|T_1| + |T_2|)$, where $|T_1|$ represents the number of pages of table T_1. The cost of construction of a node n_i is defined as a summation of the cost of its generation and storage.

(b) Transform the pivot node from the hypergraph to the oriented graph.

(c) Remove the pivot node from the hypergraph. We mention that we were inspired by the work proposed by [8] to generate the UQP which ensured the scalability of our approach. Figure 8 shows an example for the transformation step of the hypergraph to an oriented graph. In the end, the join nodes having a positive benefit are selected as candidates for materialization. In this example, the two nodes $J1$ and $J2$ are selected as candidates' views.

5. **Query Scheduling:** To increase the benefit and reusing of materialized views before their dropping, we propose to reschedule the queries of the set Q_{first}. The scheduler has the following formalization:
For a given hypergraph component, this module takes as an input the set of queries of this component and their join nodes already selected as candidates for materialization. Our scheduler module aims at providing scheduled queries in a new order maximizing the net benefit of using materialized views and reducing the overall processing cost of queries. In fact, we are inspired by the work proposed in [7] showing the efficiency of their scheduling algorithms in maximizing the benefit of materialized views and improving the query processing cost. Contrary to this algorithm in which the materialized views are all dropped after their usage by the appropriate queries. In our proposal, we keep the Pivot node from each component to serve the future incoming queries. The pivot nodes represent the views having the maximal benefit for each component. To clarify our scheduling process, let us consider the hypergraph component illustrated in the figure 9 in which 4 queries are involved. In this example, two join nodes are selected as candidates for materialization: $\{J1, J_2\}$ which are written in red. The first step of our process is to order the join nodes according to their benefits. Secondly, each query is assigned to a weight calculated by summing the benefit of its used queen nodes. Finally, we order queries according to their weights.

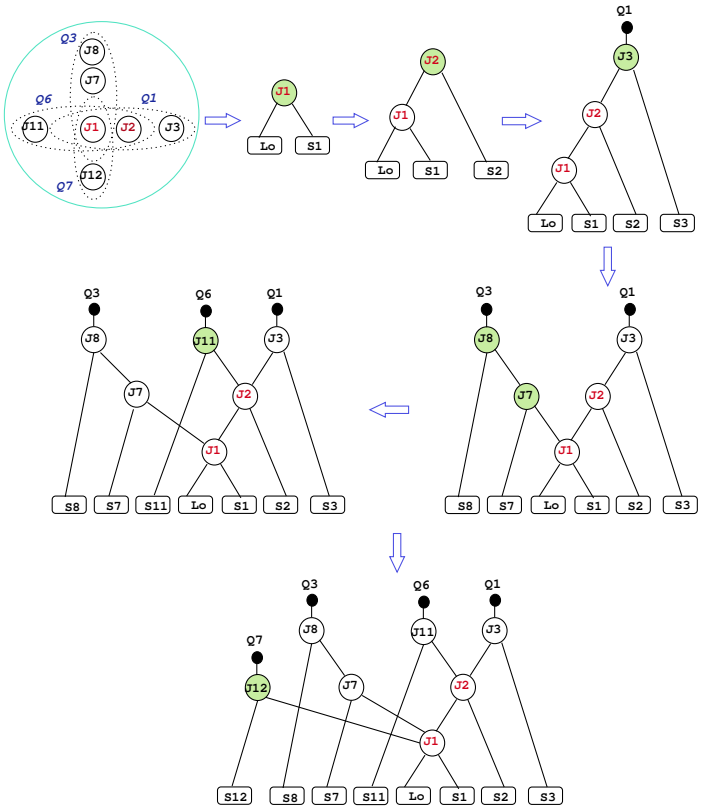


Fig. 8. From Hypergraph to Unified Query Plan

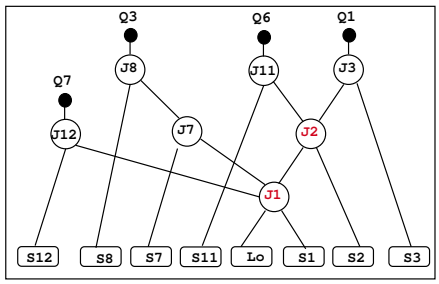
4.2. The Online Phase

In this section, we discuss our process for optimizing dynamic coming queries through our hypergraph structure. For managing the dynamic arrival of ad-hoc queries, a set of primitives are proposed for incrementally augmenting the global hypergraph constructed in the offline phase: add-node(), add-hyperedge(),remove-node(), remove-hyperedge(), etc. During the arrival of these queries, a set of materialized views is dynamically selected based on the identified shared joins through our hypergraph and a benefit function taking into account storage and maintenance constraints. At each instant t of the arrival of queries, the content and the size of the global hypergraph and the pool of materialized views will change dynamically. Two main modules characterized the online phase.

Enrichment of the Hypergraph To ensure an efficient enrichment of the global hypergraph, we put forward a strategy that consists in placing the incoming queries in the appropriate component and materializing the most shared joins identified dynamically by our hypergraph for re-using them by future queries. Our strategy for placing these queries is based on two criteria defined between the incoming query Q^t and the existing sub-hypergraphs. This is done by respecting the following principles:

Node	Benefit
J1	500
J2	100

Queries	Used Nodes	Weight
Q1	J1,J2	600
Q3	J1	100
Q6	J1,J2	600
Q7	J1	100



New Query Order	Q1	Q6	Q3	Q7

Fig. 9. Query Scheduling process

Priority 1: In order to increase the reuse of materialized views already selected and stored in the pool of each sub-hypergraph, we have defined a metric for the coming query Q^t called Nbr_Shared_Views and defined as follows .
$Nbr\_Shared\_Views(Q^t, GH_i^t) := |Joins(Q^t) \bigcap Pool^{GH_i^t}|$

The above metric represents the cardinal of the intersection among the set of join nodes of incoming query Q^t and the set of nodes already materialized for the ith sub-hypergraph GH_i^t. After calculating this metric among Q^t and each sub-hypergraph. The query Q^t has to be placed in the component that maximizes this metric.

Priority 2: If the query Q^t does not share any materialized views with the existing sub-hypergraphs, Q^t is placed in the component that shares with it the maximum of join nodes. To do that, we have defined a new metric called $Shared\ Joins\ Weight\ (Q^t, GH_i^t)$ (implemented in the algorithm2). This metric is calculated among the query Q^t and each sub-hypergraph. The metric $Shared\ Joins\ Weight$ is related to the degree of each join node belongs to Q^t in the incidence matrix of a given sub-hypergraph. The node's degree is returned from the last column of each incidence matrix that we name it FRQ column. To define the the metric $Shared\ Joins\ Weight$, we have to firstly define the degree function that allows calculating the join nodes degrees of Q^t in the incidence matrix of the ith component using algorithm 1.

Algorithm 1: Degree

Inputs a join node: V_i^t, the incidence matrix: $M^{GH_i^t}$;
Outputs the calculated degree: $Degree$;
if $V_i^t \notin nodes(GH_i^t)$ **then**
    $Degree$=0;
    /* if the vertex V_i^t does not belong to the set of vertices of the sub-hypergraph (GH_i^t), then its degree is zero */
else
    $Degree = \sum_{j=1}^{Number\_Columns-1} M^{GH_i^t}(i,j)$;
    /* We assume that the join node Vi is positioned in the ith Row */
end

Using the above degree algorithm, we have defined the algorithm $Shared\_Joins\_Weight\ (Q^t, GH_i^t)$ that calculates the weight of the incoming query Q^t in the incidence matrix of the ith hypergraph component GH_i^t.

Algorithm 2: Shared Joins Weight

Inputs The incoming query: Q^t, the ith sub-hypergraph: GH_i^t;
Outputs the calculated weight: $Shared\_Joins\_Weight$;
$Shared\_Joins\_Weight \leftarrow 0$;
foreach $node \in join\_nodes(Q^t)$ **do**
    $Degree \leftarrow Degree(node, M^{GH_i^t})$;
    if $Degree \neq 0$ **then**
        $Shared\_Joins\_Weight \leftarrow Shared\_Joins\_Weight + 1$;
    end
end

The dynamic changes in the hypergraph components impose us to dynamically updating their incidence matrix. To do so, we have defined the algorithm Update Incidence Matrix $(Q^t, M^{GH_i^t})$ that allows adding columns, rows and updating the FRQ column which represents the degree of nodes in the incidence matrix. We perform this task when a new query is placed in the ith hypergraph component. If the query Q^t does not share any join with the existing components, a new hypergraph component is constructed and associated with this query.

Algorithm 3: Update Incidence Matrix

Inputs The incoming query : Q^t , the incidence matrix: $M^{GH_i^t}$;
Outputs The updated incidence matrix: $M^{GH_i^t}$;
add the column Q^t to the matrix $M^{GH_i^t}$;
foreach $node \notin nodes(Q^t)$ **do**
    if $node \notin nodes(GH_i^t)$ **then**
        add new row to the matrix $M^{GH_i^t}$;
        $Degree(node, M^{GH_i^t}) = 1$;
    else
        $Degree(node, M^{GH_i^t}) = Degree(node, M^{GH_i^t}) + 1$;
    end
end

Dynamic Re-Selection of Materialized Views In fact, the dynamic and continuous arrival of queries produces a massive number of views to materialize which violates the storage space constraint. Indeed, it is impossible to materialize all candidate views under such a situation. To cope with this problem, we put forward a strategy that consists in materializing the most shared joins that can be used by future incoming queries. To do that, we repeat the following process at each arrival of a new query to a component until the saturation of the fixed storage space: At the arrival of a new query Q^t to the ith hypergraph component GH_i^t, we calculate the benefit of each joins belonging to the query

Q^t using our benefits function 2. Afterward, we check if there are joins having a positive benefit. In this case, we call the algorithm $Update\ Materialized\ Views$ to update the pool of the ith component GH_i^t by materializing joins which have a positive benefit. If the storage space is saturated, we drop the materialized views having the least benefit in the pool of this component and we replace them by materializing beneficial joins of the current query.

Algorithm 4: Update Materialized Views

Inputs the query: Q^t; the hypergraph component : Fi;
the views pool associated to the Component Fi: $Pool^{F_i}$; disk Space: DS;
Output the updated pool of the component Fi: $Pool^{F_i}$;
$joins \leftarrow Join(Q^t)$;
Get_Benefit (Joins);
$L \leftarrow Return\_Joins\_with\_Positive\_Benefit(Joins)$;
Descending Order(L) ;
foreach $node \in L$ **do**
  if $node \notin Pool^{F_i}$ $And\ size(Pool^{F_i}) + size(node) < DS$ **then**
    Materializing (node);
    $Pool^{F_i}.add(node)$;
    $size(Pool^{F_i}) \leftarrow size(Pool^{F_i}) + size(node)$;
  else
    if $node \notin Pool^{F_i}$ $And\ size(Pool^{F_i}) + size(node) > DS$ **then**
      Ascending Order($Pool^{F_i}$) ;
      $idx \leftarrow 0$;
      repeat
        if $benefit(node) > benefit(Pool[idx])$ **then**
          Dropping (Pool [idx]) ;
          $size(Pool^{F_i}) \leftarrow size(Pool^{F_i}) - size(node)$;
          $idx \leftarrow idx + 1$;
        end
      until $Pool^{compi}) + size(node) < DS$ *OR* $benefit(node) < benefit(Pool[idx])$;
      if $size(Pool^{compi}) + size(node) < DS$ **then**
        Materializing (node);
        $Pool^{F_i}.add(node)$;
        $size(Pool^{F_i}) \leftarrow size(Pool^{F_i}) + size(node)$;
      end
    end
  end
end

Algorithm 5 describes our dynamic process for constructing the hypergraphs and selecting materialized views in the online phase. To illustrate our dynamic strategy for managing ad-hoc and dynamic queries, an example is given in the figure 10 by considering the same query workload and hypergraph considered in the previous examples 2.

Algorithm 5: Incremental Construction of the hypergraph And Dynamic Materialization

Inputs: the incoming query Q^t at the instant t; Storage space Sp;
Outputs: a list of components (F) of GH; pool of views for each component;
loop;
$t := 1$;
Query_Parser (Q^t);
if $|F^t| = 0$ **then**
    $F^t := Construct\_new\_component()$;
    $add\_edge(F^t, Q^t)$;
    $M^t := Calculate\_incidence\_matrix(GH^t)$;
else
    foreach $F_i^t \in GH^t$ **do**
        $Nbr\_shared\_views^t := |Joins(Q^t) \bigcap Pool^{F_i^t}|$;
        $List_1^t.add(Nbr\_shared\_views^t)$;
        $Shared\_Joins\_Weight^t := Shared\ Joins\ Weight(Q^t, F_i^t)$;
        $List_2^t.add(Shared\_Joins\_Weight^t)$;
    end
    $Maximum_1^t := Maximum(List_1^t)$;
    if $Maximum_1^t = 0$ **then**
        $Maximum_2^t := Maximum(List_2^t)$;
        if $Maximum_2^t = 0$ **then**
            $Construct\_new\_component(F_i^t)$;
            $add\_hyperedge(F_i^t, Q^t)$;
            $F.add(F_i^t)$;
            $M^t := Update\_incidence\_matrix$;
        else
            $pos^t := Return\_pos(Maximum_2^t, List_2^t)$;
            $add\_hyperedge(F.get(pos^t), Q^t)$;
            $UpdateIncidenceMatrix(Q^t, M^t)$;
            $UpdateMaterializedViews(Q^t, F.get(Pos^t), Pool^{F.get(Pos^t)}, Sp)$;
        end
    else
        $Pos^t := Return\_posn(Maximum_1^t, List_1^t)$;
        $add\_hyperedge(F.get(pos^t), Q^t)$;
        $Rewrite(Q^t, Pool^{F.get(Pos^t)})$;
        $UpdateIncidenceMatrix(Q^t, M^t)$;
        $Update\ Materialized\ Views(Q^t, F.get(Pos^t), Pool^{F.get(Pos^t)}, Sp)$;
    end
end
$t := t + 1$;
END LOOP

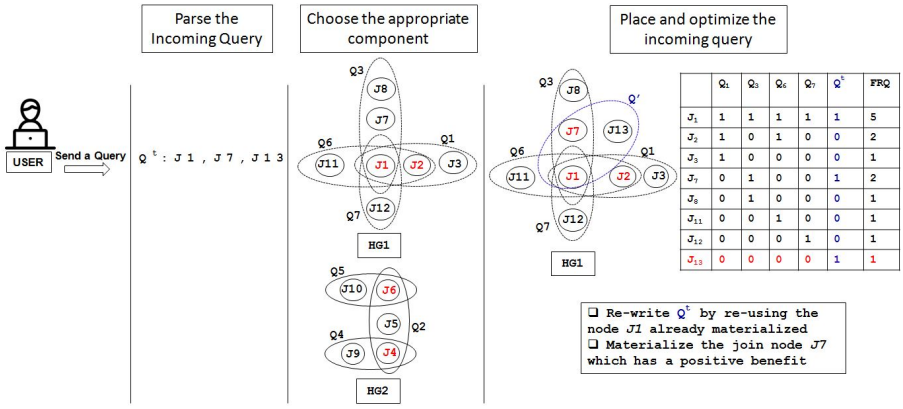


Fig. 10. The optimization process of an ad-hoc query

5. Experimental Study

In this section, we firstly show the connection of our approach to a commercial DBMS. Afterward, we conduct an efficiency study to validate and compare our proposal with major state-of-art studies.

5.1. ProRes Connection to Oracle DBMS

Based on our findings, we have developed a tool, called ProRes inspired by the well-known commercial advisors allows assisting DBA in their tasks when selecting materialized views with a strong advantage in managing the three-properties of analytical queries. ProRes is developed using Java and integrates all phases and modules of our approach. Our wizard is illustrated in fig 11.

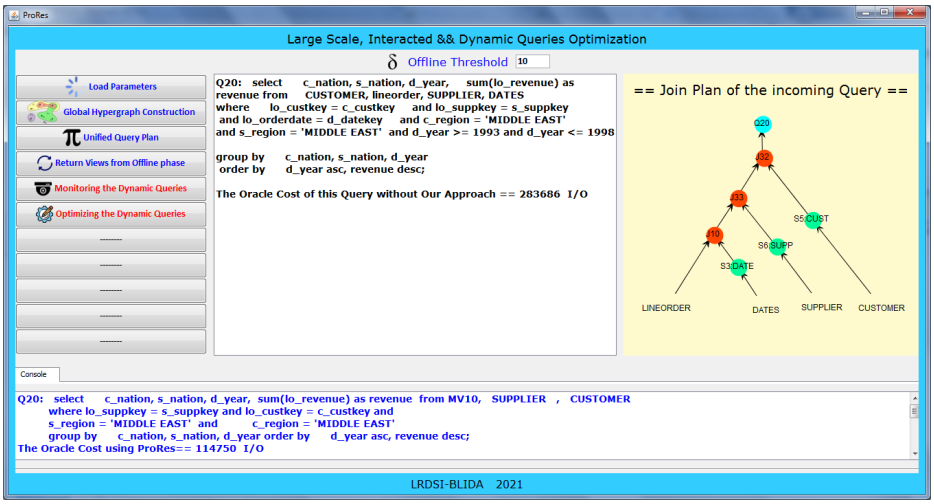


Fig. 11. An Example of Functioning of ProRes.

5.2. Efficiency Study

In this section, we present an experimental validation of our approach using the following environment: a server with E5-2690V2, 3 GHz processor, 24 GB of main memory, and 1 TB of the hard disk. We generate a DW with 30 GB deployed in Oracle 12c DBMS and queries using SSB generator modules.

Our approach has been compared against two approaches: **(1)** the approach proposed in [60] (that we name it *YANG*) which is considered the pioneer study in data warehouses that highlights the strong dependency among PMQO and VSP. For *YANG* we have developed both of their algorithms :(a) their naive algorithm called A feasible solution that generates all possible MVPP and choose the plan with minimum cost (b) their algorithm based on 0-1 integer programming which is faster than the first. **(2)** the algorithm proposed in [49] which is considered as one of the most important works that highlighted the crucial role that query scheduling plays in dealing efficiently with dynamic materialized views selection. We reference this work in our experiments by *PHAN*. For *PHAN*, we have developed the following algorithms : (a) their genetic algorithm which aims to find the optimal order of queries by using natural selection taken from Darwin's theory with 1000 generations. (b) an algorithm for selecting the nodes having the greater benefit as candidates for materialization (in [49], these nodes are selected using DB2 advisor) (c) their algorithm for pruning the set of candidates nodes based on their benefit (d) their algorithm for evaluating the net benefit of the pruned set of candidates views in optimizing a given query workload. (e) and finally, an algorithm for managing the cache following LRU rules.

Number of selected materialized views and their benefit Optimizing a big workload of queries by selecting a small set of materialized views is one of the crucial quality metrics recently highlighted by a leading DBMS editor [2]. Therefore , we attempt in this experiment to evaluate the three algorithms in terms of the number of selected materialized views, the number of optimized queries, and the number of dropped views. To do so, two experiments were conducted by considering two different workloads with 100 and 1000 queries. The obtained results are summarized in Fig.12. The obtained results show that our approach outperforms the other approaches in terms of the number of optimized queries.

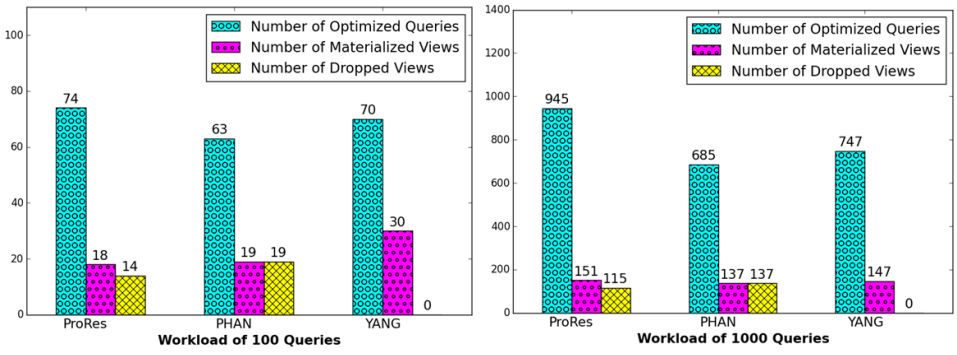


Fig. 12. A comparison among Our Approach, Phan and Yang algorithms

Impact of selected views on query processing and materialization costs The goal of this experiment is to study the contributions of the selected materialized views by our approach and the other algorithms on the overall query processing and their materialization costs. To do so, we have evaluated theoretically the different costs using our mathematical cost model. The obtained results are reported in Fig. 13. The selected views by our approach are more beneficial than those generated by the other algorithms. This is due to our materialization strategy that selects the most beneficial candidates and to our scheduling policy that allows augmenting the benefit of the selected materialized views.

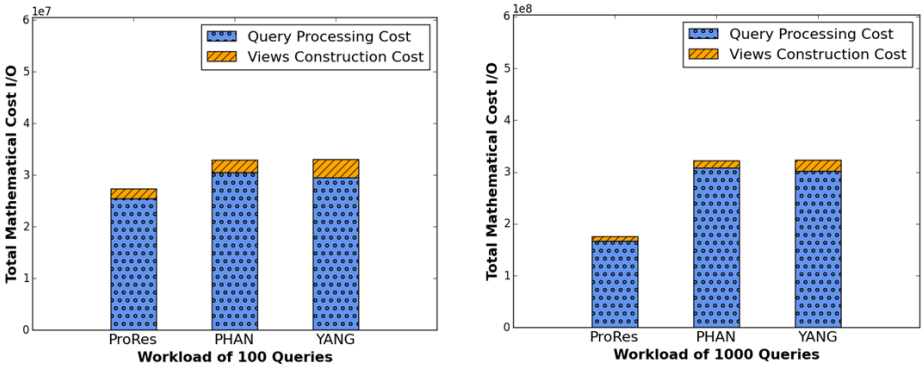


Fig. 13. Comparison among the three approaches in terms of processing/maintenance costs

Oracle Validation: Comparison of the three approaches The goal of this experiment, is to validate the results obtained theoretically in the previous experiments. To do this, we consider a workload of 100 queries running on a DW with 30 GB deployed in Oracle 12c DBMS. The storage space for materialized views is set to 60 GB. The obtained results are reported in Fig. 14. We observe that our algorithm outperforms the other algorithms. The obtained results coincide with those obtained theoretically, which confirms the efficiency and the superiority of our approach .

Impact of query scheduling and dynamic materialization on optimizing queries For testing the efficiency of our approach two scenarios are considered:

(a) Static Materialization: In this experiment, we consider a naive scenario in which the nodes selected by each algorithm are materialized till the saturation of the storage space. To perform this experiment, we have considered a data warehouse with 30 Gb and a workload of 100 queries using SSB Benchmark. As shown in Fig.15a, there is not a big difference between our approach and Yang's algorithm, which demonstrate that our approach does not avoid the selection of the best-materialized views.

(b) Dynamic Materialization with Query Scheduling: To perform this experiment, we have used the same above data by considering SSB data set with 30 Gb and a query workload of 100 queries generated randomly using the SSB generator. As shown in Fig.15b, our approach outperforms largely Phan's algorithm. This is due to the minimal

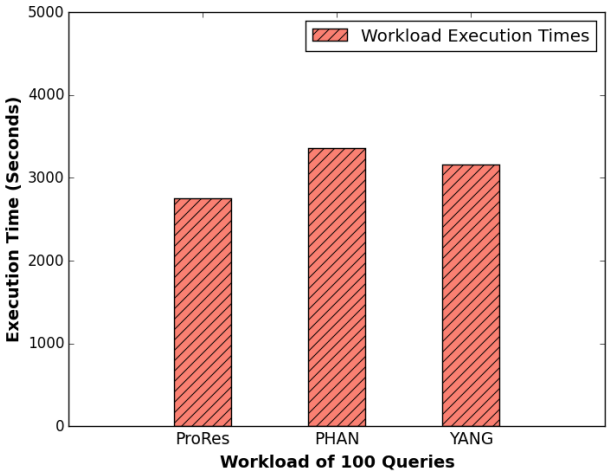
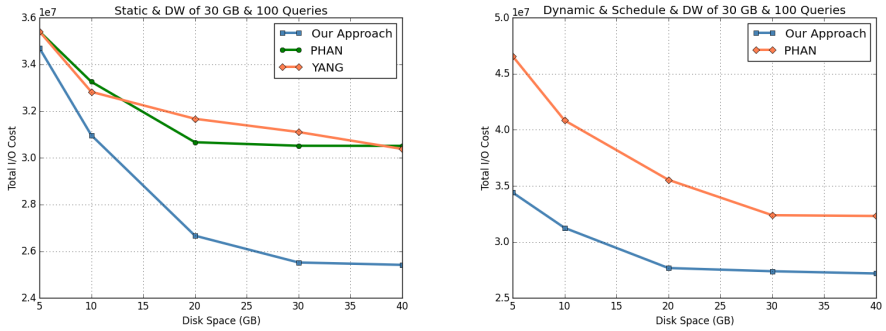


Fig. 14. Workload execution times.

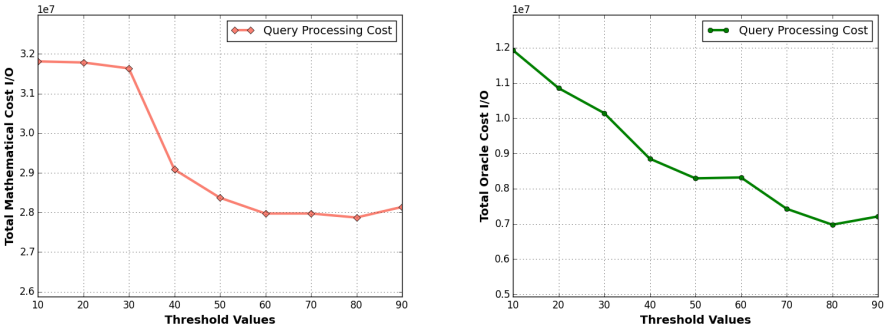
number of dropping in our approach than that of Phan. In addition, the materialized views selected by our approach are used maximally to optimize the appropriate queries before their dropping.



(a) Static scenarios **(b)** Dynamic scenarios

Fig. 15. Performance of our approach in static and dynamic scenarios

Impact of the threshold δ on the Processing Cost of Queries In this experiment, we first evaluate theoretically (using our cost model) then in Oracle 12c the overall processing cost of a given workload of 100 queries by varying the value of δ threshold. The obtained results are depicted in 16. We observe the obvious effect of the threshold δ on the query processing cost, where there is an inverse relationship between the threshold values δ and the processing cost of queries. At each increase in the threshold value, the processing cost of queries decreased until the threshold value reaches 70 which represents the stability point of the processing cost optimization.



(a) From theoretical Perspective **(b)** Oracle Validation

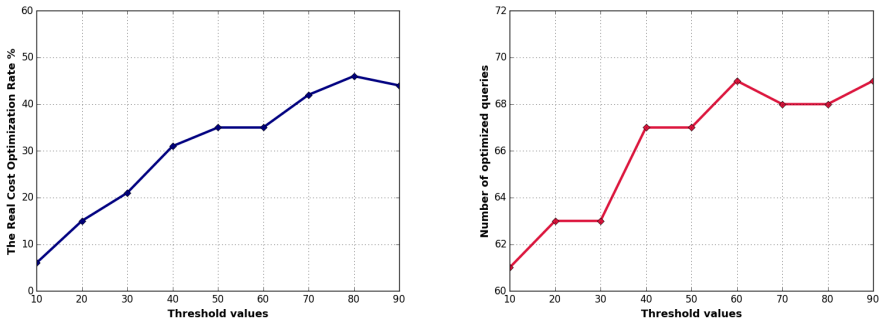
Fig. 16. The Effect of the Threshold on the Processing Cost

The processing cost reduction rate according to the threshold values In this experiment, we follow the same above scenario by considering the same set of 100 queries and varying the δ threshold values. The goal of this experiment is to evaluate the impact of the threshold values and the quality of the materialized views selected by our proactive strategy on the overall processing cost of queries. To do this, we firstly estimate the overall real processing cost of the query workload without using our approach (costwithout). Afterward, we estimate the processing cost of this workload using our approach by varying the offline threshold values. Finally, we compute the cost reduction rate as : $1 - \frac{query\ cost\ with\ views}{query\ cost\ without\ views}$.
Fig.17a shows the obtained results implemented in Oracle 12c DBMS. The obtained results confirm the previous results and prove that our approach becomes more interesting when the offline threshold increases until the stability point value $\delta \geq 70$, where the cost reduction rate is among 42% and 47%. This is due to the expansion of our pool by the most beneficial materialized views.

The Number of optimized queries according to the offline threshold values: In the last experiment, we attempt to evaluate our proactive strategy in terms of the number of optimized queries according the threshold values δ . To do this, we consider the same above scenarios and the same workload of queries. The obtained results are described in Fig17b. They showed the proportional Relationship among the number of optimized queries and the offline threshold values until the value $\delta \geq 60$, where there is a stability in the number of optimized queries, where 69 queries have been optimized from the 100 received queries.

6. Conclusion

In this paper, we discuss the opportunity offered by the best aspects brought by the era of Big Data to augment the data warehouses technology. Note that several efforts have been recently deployed to augment this technology. But, they did not give great attention to the high number of queries as the other aspects despite its strong connection to the different phases of the data warehouse life cycle. Today, analytical queries are known by three



(a) The Oracle cost reduction rate **(b)** The Number of optimized queries

Fig. 17. The cost reduction rate and the Number of optimized queries according to the offline threshold

main properties: (1) very numerous, (2) dynamic, and (3) share similar operations. These characteristics may touch several well-studied and known problems such as multi-query optimization (PMQO), physical design. The classical solutions cannot be used directly to handle these three properties during the physical and logical optimization of queries. We guess that dealing with these three properties requires the usage of a flexible data structure. Therefore, hypergraphs have been proposed for coupling PMQO and VSP under a new angle by considering our three properties. In this work, we were inspired by the aspects of new business intelligence (BI next-generation), where there are two types of queries: a priori known queries and ad-hoc queries. To deal with both types of queries, we have proposed a proactive approach composed of two phases: an offline phase for optimizing queries known in advance and an online phase dedicated to optimizing the ad-hoc queries. The offline phase and the online phase share the use of the same hypergraph structure for capturing the query interaction and selecting the candidate views. The main particularity of our approach is that it covers the two main categories of the dynamic selection of views. Our proposal implemented by a simulator called ProRes was validated theoretically using mathematical cost models and its results were directly implemented on the Oracle DBMS. The obtained results are encouraging and show the efficiency and effectiveness of our approach.

Our work opens several challenges: (i) currently, we are integrating our proposal into PostgreSQL DBMS, (ii) considering other optimization techniques such as horizontal data partitioning and indexes, (iii) integrating the machine learning techniques for predicting the appropriate optimization structure for the ad-hoc queries. (iv) reproduce our proposal for using hypergraphs to deal with SPARQL queries in semantic databases.

References

1. Agrawal, S., Chaudhuri, S., Kollár, L., Marathe, A.P., Narasayya, V.R., Syamala, M.: Database tuning advisor for microsoft SQL server 2005. In: VLDB. pp. 1110–1121 (2004)
2. Ahmed, R., Bello, R.G., Witkowski, A., Kumar, P.: Automated generation of materialized views in oracle. Proc. VLDB Endow. 13(12), 3046–3058 (2020)

3. Azgomi, H., Sohrabi, M.K.: A game theory based framework for materialized view selection in data warehouses. Engineering Applications of Artificial Intelligence pp. 125–137 (2018)
4. Bellatreche, L., Karlapalem, K., Schneider, M.: On efficient storage space distribution among materialized views and indices in data warehousing environments. In: ACM CIKM. pp. 397–404 (2000)
5. Bellatreche, L., Kerkad, A.: Query interaction based approach for horizontal data partitioning. IJDWM 11(2), 44–61 (2015)
6. Boukorca, A.: Hypergraphs in the Service of Very Large Scale Query Optimization. Application. Phd thesis, ISAE-ENSMA, Poitiers France (2016)
7. Boukorca, A., Bellatreche, L., Cuzzocrea, A.: SLEMAS: an approach for selecting materialized views under query scheduling constraints. In: International Conference on Management of Data (COMAD) . pp. 66–73 (2014)
8. Boukorca, A., Bellatreche, L., Senouci, S.B., Faget, Z.: Coupling materialized view selection to multi query optimization: Hyper graph approach. IJDWM 11(2), 62–84 (2015)
9. Bretto, A.: Hypergraph Theory: An Introduction. Springer (2013)
10. Bruno, N., Jain, S., Zhou, J.: Continuous cloud-scale query optimization and processing. Proc. VLDB Endow. 6(11), 961–972 (2013)
11. de Carvalho Costa, R.L., Moreira, J., Pintor, P., dos Santos, V., Lifschitz, S.: Data-driven performance tuning for big data analytics platforms. Big Data Research pp. 100–206 (2021)
12. Çatalyürek, Ü.V., Aykanat, C.: Patoh (partitioning tool for hypergraphs). In: Padua, D.A. (ed.) Encyclopedia of Parallel Computing, pp. 1479–1487. Springer (2011), https://doi.org/10.1007/978-0-387-09766-4_93
13. Chakravarthy, U.S., Minker, J.: Multiple query processing in deductive databases using query graphs. In: VLDB. pp. 384–391 (1986)
14. Chaudhuri, S., Narasayya, V.R.: Self-tuning database systems: A decade of progress. In: VLDB. pp. 3–14 (2007)
15. Chen, T., Narita, K.: Multiple query optimization in sql-on-hadoop systems. US Patent 10,572,478 (2020)
16. Cosar, A., Lim, E.P., Srivastava, J.: Multiple query optimization with depth-first branch-and-bound and dynamic query ordering. In: International Conference on Information and Knowledge Management(ACM-CIKM). pp. 433–438 (1993)
17. Curino, C., Zhang, Y., Jones, E.P.C., Madden, S.: Schism: a workload-driven approach to database replication and partitioning. PVLDB 3(1), 48–57 (2010)
18. Dobra, A., Garofalakis, M.N., Gehrke, J., Rastogi, R.: Sketch-based multi-query processing over data streams. Data Stream Management pp. 241–261 (2016)
19. Dobra, A., Garofalakis, M.N., Gehrke, J., Rastogi, R.: Multiple-query optimization of regular path queries. In: International Conference on Data Engineering (ICDE). pp. 1426–1430 (2017)
20. Fan, W., Yu, J.X., Li, J., Ding, B., Qin, L.: Query translation from xpath to SQL in the presence of recursive dtds. VLDB Journal 18(4), 857–883 (2009)
21. Fuentes, J., Sáez, P., Gutierrez, G., Scherson, I.D.: A method to find functional dependencies through refutations and duality of hypergraphs. Computer Journal 58(5), 1186–1198 (2015)
22. Goasdoué, F., Karanasos, K., Leblay, J., Manolescu, I.: View selection in semantic web databases. Proc. VLDB Endow. 5(2), 97–108 (2011)
23. Gupta, A., Sudarshan, S., Viswanathan, S.: Query scheduling in multi query optimization. In: The International Database Engineering And Applications Symposium (IDEAS). pp. 11–19 (2001)
24. Gupta, H., Mumick, I.S.: Selection of views to materialize under a maintenance cost constraint. In: The International Conference on Database Theory (ICDT). pp. 453–470 (1999)
25. Harinarayan, V., Rajaraman, A., Ullman, J.D.: Implementing data cubes efficiently. In: The ACM Special Interest Group on Management of Data (ACM-SIGMOD). pp. 205–216 (1996)

26. Ioannidis, Y.E., Kang, Y.C.: Left-deep vs. bushy trees: An analysis of strategy spaces and its implications for query optimization. In: The ACM Special Interest Group on Management of Data (ACM-SIGMOD). pp. 168–177 (1991)
27. Jiang, W., Qi, J., Yu, J.X., Huang, J., Zhang, R.: Hyperx: A scalable hypergraph framework. IEEE Trans. Knowl. Data Eng. 31(5), 909–922 (2019)
28. Jin, C., Carbonell, J.G.: Predicate indexing for incremental multi-query optimization. In: The International Symposium on Methodologies for Intelligent Systems (ISMIS) . pp. 339–350 (2008)
29. Kalnis, P., Mamoulis, N., Papadias, D.: View selection using randomized search. Data and Knowledge Engineering 42(1), 89–111 (2002)
30. Karypis, G., Aggarwal, R., Kumar, V., Shekhar, S.: Multilevel hypergraph partitioning: Application in vlsi domain. In: The Design Automation Conference (DAC). pp. 526–529 (1997)
31. Karypis, G., Aggarwal, R., Kumar, V., Shekhar, S.: Multilevel hypergraph partitioning: applications in vlsi domain. IEEE Trans. Very Large Scale Integr. Syst. 7(1), 69–79 (1999)
32. Kementsietsidis, A., Neven, F., de Craen, D.V., Vansummeren, S.: Scalable multi-query optimization for exploratory queries over federated scientific databases. PVLDB pp. 16–27 (2008)
33. Kotidis, Y., Roussopoulos, N.: Dynamat: A dynamic view management system for data warehouses. In: The ACM Special Interest Group on Management of Data (ACM-SIGMOD). pp. 371–382 (1999)
34. Lanasri, D., Khouri, S., Bellatreche, L.: Trust-aware curation of linked open data logs. In: The INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING (ER). pp. 604–614 (2020)
35. Larson, P., Yang, H.Z.: Computing queries from derived relations. In: The International Conference on Very Large Data Bases (VLDB). pp. 259–269 (1985)
36. Le, W., Kementsietsidis, A., Duan, S., Li, F.: Scalable multi-query optimization for sparql. In: The International Conference on Data Engineering (ICDE). pp. 666–677 (2012)
37. Liang, X., Elmore, A.J., Krishnan, S.: Opportunistic view materialization with deep reinforcement learning. CoRR abs/1903.01363 (2019), http://arxiv.org/abs/1903.01363
38. Liu, L., Özsu, M.T. (eds.): Encyclopedia of Database Systems, 2nd Edition. Springer (2018)
39. Maier, C., Dash, D., Alagiannis, I., Ailamaki, A., Heinis, T.: PARINDA: an interactive physical designer for postgresql. In: The International Conference on Extending Database Technology (EDBT). pp. 701–704
40. Mami, I., Bellahsene, Z.: A survey of view selection methods. SIGMOD Rec. 41(1), 20–29 (2012)
41. Marroquin, R., Müller, I., Makreshanski, D., Alonso, G.: Pay one, get hundreds for free: Reducing cloud costs through shared query execution. In: ACM Symposium on Cloud Computing. pp. 439–450 (2018)
42. Michiardi, P., Carra, D., Migliorini, S.: Cache-based multi-query optimization for data-intensive scalable computing frameworks. Inf. Syst. Frontiers 23(1), 35–51 (2021)
43. Mistry, H., Roy, P., Sudarshan, S., Ramamritham, K.: Materialized view selection and maintenance using multi-query optimization. In: The ACM Special Interest Group on Management of Data (ACM-SIGMOD). pp. 307–318 (2001)
44. Monika Rokosik, M.W.: Efficient processing of streams of frequent itemset queries. In: The European Conference on Advances in Databases and Information Systems (ADBIS). pp. 15–26 (2014)
45. Mouna, M.C., Bellatreche, L., Narhimene, B.: HYRAQ: optimizing large-scale analytical queries through dynamic hypergraphs. In: IDEAS 2020: 24th International Database Engineering & Applications Symposium, Seoul, Republic of Korea, August 12-14, 2020. pp. 17:1–17:10. ACM (2020), https://dl.acm.org/doi/10.1145/3410566.3410582
46. O'Gorman, K., Agrawal, D., Abbadi, A.E.: Multiple query optimization by cache-aware middleware using query teamwork. In: The International Conference on Data Engineering (ICDE). p. 274 (2002)

47. Pavlo, A., Butrovich, M., Joshi, A., Ma, L., Menon, P., Aken, D.V., Lee, L., Salakhutdinov, R.: External vs. internal: An essay on machine learning agents for autonomous database management systems. IEEE Data Eng. Bull. 42(2), 32–46 (2019)
48. Perez, L.L., Jermaine, C.M.: History-aware query optimization with materialized intermediate views. In: The International Conference on Data Engineering (ICDE). pp. 520–531 (2014)
49. Phan, T., Li, W.: Dynamic materialization of query views for data warehouse workloads. In: The International Conference on Data Engineering (ICDE). pp. 436–445 (2008)
50. Rehrmann, R., Binnig, C., Böhm, A., Kim, K., Lehner, W., Rizk, A.: Oltpshare: The case for sharing in OLTP workloads. Proc. VLDB Endow. 11(12), 1769–1780 (2018)
51. Roukh, A., Bellatreche, L., Bouarar, S., Boukorca, A.: Eco-physic: Eco-physical design initiative for very large databases. Information Systems pp. 44–63 (2017)
52. Roy, P., Sudarshan, S.: Multi-query optimization. In: In [38] (2018), https://doi.org/10.1007/978-1-4614-8265-9_239
53. Savva, F., Anagnostopoulos, C., Triantafillou, P.: Adaptive learning of aggregate analytics under dynamic workloads. Future Gener. Comput. Syst. 109, 317–330 (2020)
54. Scheuermann, P., Shim, J., Vingralek, R.: WATCHMAN : A data warehouse intelligent cache manager. In: The International Conference on Very Large Data Bases (VLDB). pp. 51–62 (1996)
55. Schlag, S.: High-Quality Hypergraph Partitioning. Ph.D. thesis, Karlsruhe Institute of Technology, Germany (2020), https://nbn-resolving.org/urn:nbn:de:101:1-20200304035816201657 65
56. Sellis, T.K.: Multiple-query optimization. ACM Trans. Database Syst. 13(1), 23–52 (1988)
57. Shim, K., Sellis, T.K., Nau, D.S.: Improvements on a heuristic algorithm for multiple-query optimization. Data Knowl. Eng. 12(2), 197–222 (1994)
58. Tapdiya, A., Xue, Y., Fabbri, D.: A comparative analysis of materialized views selection and concurrency control mechanisms in nosql databases. In: IEEE International Conference on Cluster Computing (CLUSTER). pp. 384–388 (2017)
59. Timos K. Sellis, S.G.: On the multiple query optimization problem. IEEE Transactions on Knowledge and Data Engineering pp. 262–266 (1990)
60. Yang, J., Karlapalem, K., Li, Q.: Algorithms for materialized view design in data warehousing environment. In: The International Conference on Very Large Data Bases (VLDB). pp. 136–145 (1997)
61. Yu, J.X., Yao, X., Choi, C.H., Gou, G.: Materialized view selection as constrained evolutionary optimization. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 33(4), 458–467 (2003)
62. Yu, X., Li, G., Chai, C., Tang, N.: Reinforcement learning with tree-lstm for join order selection. In: The International Conference on Data Engineering (ICDE). pp. 1297–1308 (2020)
63. Zdonik, S.B., Maier, D. (eds.): Readings in Object-Oriented Database Systems. Morgan Kaufmann (1990)
64. Zilio, D.C., Rao, J., Lightstone, S., Lohman, G.M., Storm, A.J., Garcia-Arellano, C., Fadden, S.: DB2 design advisor: Integrated automatic physical database design. In: The International Conference on Very Large Data Bases (VLDB). pp. 1087–1097 (2004)
65. Zlatic, V., Ghoshal, G., Caldarelli, G.: Hypergraph topological quantities for tagged social networks. CoRR abs/0905.0976 (2009)

Appendix

```
Q1: select c_nation,s_nation,d_year,sum(lo_revenue) as revenue
from     CUSTOMER, lineorder, SUPPLIER, DATES
where  lo_custkey = c_custkey
and lo_suppkey = s_suppkey
and lo_orderdate = d_datekey
```

```
and c_region = 'AFRICA'
and s_region = 'AFRICA'
and d_year >= 1992 and d_year <= 1997
group by c_nation, s_nation, d_year
order by d_year asc, revenue desc;


Q2: select c_city,s_city,d_year,sum(lo_revenue) as revenue
from     CUSTOMER, lineorder, SUPPLIER, DATES
where lo_custkey = c_custkey
and lo_suppkey = s_suppkey
and lo_orderdate = d_datekey
and c_nation = 'VIETNAM'
and s_nation = 'VIETNAM'
and d_year >= 1993 and d_year <= 1998
group by c_city, s_city, d_year
order by d_year asc, revenue desc;

Q3:select c_city,s_city,d_year,sum(lo_revenue) as revenue
from     CUSTOMER, lineorder, SUPPLIER, DATES
where  lo_custkey = c_custkey
and lo_suppkey = s_suppkey
and lo_orderdate = d_datekey
and (c_city='IRAQ     4' or c_city='JORDAN   6')
and (s_city='IRAQ     4' or s_city='JORDAN   6')
and d_year >= 1992 and d_year <= 1997
group by c_city, s_city, d_year
order by d_year asc, revenue desc;

Q4:select c_city,d_year,sum(lo_revenue) as revenue
from     CUSTOMER, lineorder, DATES
where lo_custkey = c_custkey
and lo_orderdate = d_datekey
and c_nation = 'GERMANY'
and d_year >= 1993 and d_year <= 1998
group by c_city, d_year
order by d_year asc, revenue desc;

Q5: select s_city,p_brand,sum(lo_revenue-lo_supplycost) as profit
from     SUPPLIER,lineorder, PART
where  lo_suppkey = s_suppkey
and lo_PARTkey = p_PARTkey
and s_nation = 'VIETNAM'
and p_category = 'MFGR#34'
group by  s_city, p_brand
order by  s_city, p_brand;

Q6: select c_nation,s_nation,d_year,sum(lo_revenue) as revenue
from     CUSTOMER, lineorder, SUPPLIER, DATES
where lo_custkey = c_custkey
and lo_suppkey = s_suppkey
and lo_orderdate = d_datekey
and c_region = 'AFRICA'
and s_region = 'ASIA'
and d_year >= 1992 and d_year <= 1997
group by c_nation, s_nation, d_year
order by d_year asc, revenue desc;

Q7:select  c_city,d_year,sum(lo_revenue) as revenue
from     CUSTOMER, lineorder, DATES
where  lo_custkey = c_custkey
and lo_orderdate = d_datekey
and c_nation = 'BRAZIL'
and d_year >= 1992 and d_year <= 1997
group by  c_city, d_year
order by  d_year asc, revenue desc;
```

Mustapha Chaba Mouna is a last-year PhD student at University Blida 1, Blida, Algeria. He holds a master's degree in Computer Science from the same university. His research interest focuses on query processing and optimization.

Ladjel Bellatreche is a Full Professor at National Engineering School for Mechanics and Aerotechnics (ISAE-ENSMA), Poitiers, France. He leads the Data and Model Engineering Team of the Laboratory of Computer Science and Automatic Control for Systems (LIAS). He is also a Part Time Professor at Harbin Institute of Technology (HIT) since 2019. His was a visiting professor of the Québec en Outaouais - Canada (2009), a Visiting Researcher at Purdue University - USA (2001) and Hong Kong University of Science and Technology, China (1997-1999). His research interest focuses on Data Management Systems and Semantic Web. He serves as an Associate Editor of the Data & Knowledge (DKE) Journal, Elsevier. His research projects have been funded by 2 EU projects.

Narhimene Boustia is a full Professor at university Blida 1, Blida, Algeria, where she joined as a faculty member since Oct. 2002. Her research interest focuses on data security, access control and data Management. She has co-authored more than 30 papers and received more than 90 citations (H-index=6).

Received: June 06, 2021; Accepted: September 15, 2021.