DAMS: Dynamic Association for View Materialization Based on Rule Mining Scheme

Ashwin Verma, Pronaya Bhattacharya, Umesh Bodkhe, Akhilesh Ladha, and Sudeep Tanwar

Abstract In data warehousing, view selection (VS) is an important aspect. Optimal VS needs to be materialized in order to minimize the overall data retrieval time. To support the same, performance metrics like memory constraints to save materialized views, query execution time, and query workloads needs to be addressed to reduce the overall retrieval time. As far as static view materialization (VM) is concerned, pre-computing strategies are required to execute the query workload prior to VM, but the approach is not scalable for small disk sizes. In the current era, the memory requirement is humongous to store pre-computed views in the materialized query table (MQT) that adds an overhead to view maintenance cost and disk sizes. To address the aforementioned issues, the authors propose a novel VM scheme *DAMS*. *DAMS* operates in three phases. In the first phase, the scheme chooses a materialized view in a dynamic and on-demand basis to reduce the query processing time. Then, in the second phase, a novel attribute selection algorithm is proposed based on association rule mining (ARM) technique in VS to address historical queries. It selects a candidate view from a pool of such views. As the number of queries is large, the proposed algorithm reduces the computational latency in fetching the view result. Finally, selected views are prioritized by grouping items as clusters set based on support and confidence metrics to speed up VM operations.

A. Verma · P. Bhattacharya (✉) · U. Bodkhe · A. Ladha · S. Tanwar
Institute of Technology, Nirma University, Ahmadabad, Gujarat, India
e-mail: pronoya.bhattacharya@nirmauni.ac.in

A. Verma
e-mail: ashwin.verma@nirmauni.ac.in

U. Bodkhe
e-mail: umesh.bodkhe@nirmauni.ac.in

A. Ladha
e-mail: akhilesh.ladha@nirmauni.ac.in

S. Tanwar
e-mail: sudeep.tanwar@nirmauni.ac.in

P. K. Singh et al. (eds.), *Recent Innovations in Computing*, Lecture Notes in Electrical Engineering 701, https://doi.org/10.1007/978-981-15-8297-4_43

Keywords View materialization · View selection · Candidate views · Materialized query table · Least recently used

1 Introduction

The rise in data generation by devices has raised challenges to explore optimal ways to visualize data. Earlier, visualization is conducted through static tools like tableau, Microsoft Excel and other related applications [12]. Today, the collected datasets are humongous with multiple attributes [10]; thus, VS is a critical issue. Manual approaches to determine the most appropriate view through a list of offered represented views are tedious and time-consuming. Thus, there is a pressing need to dynamically allocate views based on on-demand query fired by users. The problems are that there are numerous factors on which a particular query is raised like similarities in view generations, informational semantics, grouping of data and aggregation. This requires a scanning of the entire MQT every time a query is fired which is cost-ineffective.

A static or dynamic approach can be used for materialization of view. Static VM materialize views before the execution of workload and remains as it is till the last statement of workload, in between if database object update the relation or tables that will not immediately propagated for the remaining workload, which create inconsistency in the database, in contrast to dynamic view will be updated automatically when the database object modifies the database, we can understand the concept of VM process in Fig. 1. Incremental approaches were proposed by authors that define multiple performance metrics like accuracy in view updates, misses in base tables and latency in fetching queries. The performance parameter includes optimizes a query sequence and involves complexity of view, memory constraints and selectivity of view [8]. A materialized view is an object of database which contains a result of the query, which may be local copy of data located remotely or may be a join result or a summary of an aggregate function. The collection of views is effectively chosen in such a way that most of the query can be answered, through which we can reduce the overall execution time of the query. The drawback of VM is that they need to be refreshed timely whenever an update happens in the base tables. Thus, the overall system is not scalable as the number of users increases. Table 1 presents a comparative analysis of different existing approaches for VM.

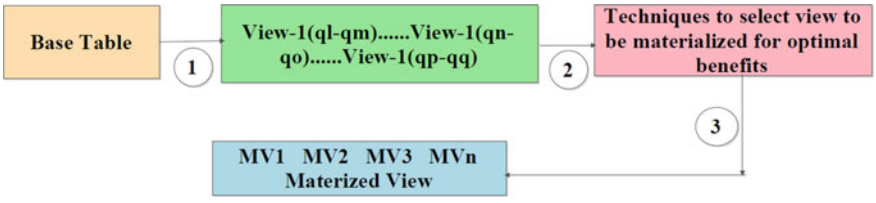

Fig. 1 Sequence flow of view materialization

Table 1 Comparative table for existing approaches for VM

Authors	Years	Objective	Pros.	Cons.
Rashid et al. [21]	2010	Reducing the maintenance cost of materialized view in object relational DBMS	Incremental model to improve query performance cost	Results limited to static view models, hence not scalable with increasing users
Anter et al. [3]	2012	A hybrid integrated system to materialized view based on past queries	Distribution of queries based on user choices	More parameters need to be considered like query execution time and query fetching time in conjunction with user choices
Zlamaniec et al. [28]	2015	Increase the SPARQL (SPARQL protocol and resource description framework) endpoint availability	Selection of view is based on the access patterns and frequency of access to support real-time experience	Optimization is limited toward data retrieval methods for SPARQL database only
Yoshifumi et al. [19]	2017	A PROforma-based approach on view updates, computation-based calculations of temporal updates	Cartesian product and join views became updatable	Not applicable on relational DBMS as view adaptability is managed by INSTEAD OF triggers
Kumar et al. [18]	2017	Materialized view based on discrete genetic operators	Particle swarm optimization (PSO) to select top k views from the multidimensional lattice for materialization	The approach is compared with heuristic and local optimum is achieved with low convergence rate during subsequent iterations
Jindal et al. [16]	2018	Expression of queries on cluster datasets to compute common sub-expressions for view materialization	Reusability of common sub-expressions to expedite the execution time of view fetch	Redundancy in different sub-expressions during clustering of datasets

(continued)

Table 1 (continued)

Authors	Years	Objective	Pros.	Cons.
Azgomi et al. [6]	2018	Game theory with players' view maintenance costs and query processing costs to decide play-off function to reach Nash equilibrium	Convergence toward equilibrium state is fast and efficient resulting in reduced query processing costs	Whenever payoff does not increase with subsequent iterations, the game reaches a local optimal equilibrium point which is far from ash convergence
Ye et al. [27]	2018	A multi-view clustering method to reduce the effects of noisy views during clustering process	Allows assignment of small weights to improve clustering performance	Focuses on clustering the relevant views, still we need to find out different attributes which will be considered for the view materialization
Gosain et al. [13]	2018	Priority-based VM scheme for data warehouse	PSO for selection of prioritized set of queries for VM on data cube	Approach is not suitable for dynamic view selections
María et al. [11]	2019	Assertions are designed with the help of materialization to ensure tuples in view follow cross-row constraints	Suited for high complex view models and large-sized queries	The model is not scalable in terms of access cost as the number of queries increases in the system

In the literature, many authors proposed solutions to address challenges in VM like scalability, information extraction and retrieving patterns from raw data. The various conditions where a view is needed to be materialized [21], materialization in hybrid integration [3] and the optimization of materialization in SPARQL are discussed [28]. PROforma-based approach for view updation [19] and use of discrete genetic operator and optimization technique is used to materialize the view [18]. Overlapping sub-expression [16] to identify the common expression for materialization and game theory [6]-based framework for selection of view is discussed. Priority-based [13] materialization method, multi-clustering [27] to remove noisy view from materialization process and cross-row constraints [11] in materialization are addressed. Data mining techniques [22], frequent pattern analysis [24], are proposed to support ARM applications like word embedding in textual data [15], extract labels in intrusion detection system [5] and decentralized storage patterns in blockchain [17]. The authors in [26] addressed the problems of selecting view dynamically and dropping of MV. Researchers in [4, 7] use a clustering method to find the cluster of closely

related queries. Phan et al. [20] describe automated, dynamic materialized query table management scheme that materializes views.

Materialized view can be managed by least recently used (LRU) policy. Hossein et al. [6] proposed a game theory approach where one player is greedy for high query processing time and the other layer is greedy for the high view maintenance cost. The main advantage of the approach is flexibility. However, if the number of parameters increases the system becomes complex to find materialized set.

1.1 Motivation

As the amount of data on the server is huge, therefore, for better performance views need to be processed and stored in an efficient manner in a systematic storage system where it can be frequently accessed based on business logics. So, to minimize the query execution time we need, we cannot bring the whole relation or database in memory, and we need to focus only on the selected portion which is sufficient to answer the maximum query workload. Many authors propose data mining [4], pattern analysis [23, 25], label extraction [5] and game theory approaches [6] to address VM issues. However, reducing the size of relevant data in memory is important. Hence, motivated by the same, *DAMS* proposes a dynamic VM approach which has twofold benefits. First, to reduce the query execution time by materializing the selected candidate views through proposed attribute selection algorithm based on association rules, and then, view prioritization is achieved based on clustering views on confidence and support metric results.

1.2 Research Contributions

The proposed research contributions are now as follows:

- A novel attribute selection algorithm is presented based on association rule mining in VS to address historical queries.
- Selected views are clustered-based support and confidence matrix and then prioritize for materialization.

1.3 Organization of the Paper

The contributed paper work is organized into five sections. Section 1 gives introduction and motivation behind the paper, and Sect. 2 presents the key terminologies in view materialization process. Section 3 confers the proposed approach and architecture. Section 4 presents the research challenges and future scope in the materialization

process and solving associated attribute sets. Finally, Sect. 5 presents the conclusion at the end.

2 View Materialization: Key Terminologies

For view materializing, we need to consider both space and processing time for a query as a constraint in the system. As the database object updates any information in the base table of the database, we need to propagate those changes in the view. Hence, the key challenge is to maintain the consistency in the materialized view. Some systems re-compute the materialized view from the scratch, which is not desirable as re-computations in base tables consume more time. The re-computations of the views need to be done in an incremental fashion.

2.1 Performance Evaluation Parameters for View Materialization

Selectivity of View: Selectivity of view can be defined as follows [21]

$$\text{View\_selectivity} = \frac{N_{\text{rowquery}}}{N_{\text{rowview}}}$$

where N_{rowquery} denotes the number of qualified rows and N_{rowview} denotes the number of rows existing in the view. The views are calculated with the help of a different parameter and constraint which results in a different view selectivity for the same database.

To understand this better consider the following example:

```
CREATE VIEW sales-view-1 AS SELECT att-1, att-2 FROM sales
WHERE att-3=k1 HAVING SUM(att-4)>k2.
```

Suppose the total number of rows in the base table is 5000 and the number of rows in the sales-view-1 is 1500 and we change the constraints to WHERE att-3 > $k1$ and att-3 < $k2$ HAVING AVG(att-4) > $k3$ GROUP BY att-5. And let us say the resulted rows in sales-view-1 become 1000, and now you can understand the difference in the view selectivity due to different constraints on the same database table.

View selectivity (sales-view-1) = 1500/5000 = 0.3

View selectivity (sales-view-1) = 1000/5000 = 0.2

View Complexity: It can be defined as a result of join between two or more relations and combination of WHERE and GROUP BY clause, because view with JOIN

another combination of clause is complex and takes time to compute. Understand the complexity of both sales-view-1 and sales-view-2.

```
CREATE VIEW sales_view-1 AS SELECT att-1, att-2 FROM sales
WHERE att-3=k1.
CREATE VIEW sales_view-2 AS SELECT att-1, att-2 FROM sales
WHERE att-3=k1 HAVING AVG(att-4)>k2 GROUP BY att-5.
```

Database Size: It depends on the organization and their business requirement, how frequently data is accessed from multiple resources for analysis and fact finding. A small organization relatively stores small size database as compared to the organization with huge amount of data stored in different geographical locations. So, the size of materialized view also depends on organization and its requirement.

Query Optimization: Based on the view calculated and stored in the materialized query table, few conditions are applied to check whether a view is capable of providing the solution or not; if it is not, we need to fetch the result from the original base table. So generally we prefer one of the two approaches, In the first approach, we match the attribute of the query with the view, or alternatively in second approach we can compare the join attribute, selection condition and aggregate function between the query from the workload and materialized view. The following is depicted through an example on defining a view.

```
CREATE VIEW Employee_Department_V1 as SELECT E.
E.employee_name, D.department_name,
D.department_location, M.employee_name as
Contractor FROM Employee E JOIN Department D us
ing(department_no) join Employee M on (M.employee_no=
D.manager);
Query: SELECT E.employee_name, D.department_name FROM Em
ployee E, Department D WHERE D.department_no=101 and
M.employee_no=D.manager;
```

A VS problem is based on parameter size of the view, frequency of view update [9], query prioritization [14] and cost function. The approaches of dynamic view materialization are depicted in Fig. 2.

3 *DAMS:* The Proposed Approach

In the proposed approach, a heuristic framework is presented. Workload is considered as a sequence of statements $\{s_1, s_2, \ldots, s_n\}$ to be executed in a specific order. Figure 3 shows the VM process. For view selection, mainly four steps are involved:

1. Preprocessing of queries to generate attribute matrix
2. Identify the association rules
3. Identify clusters
4. View selection.

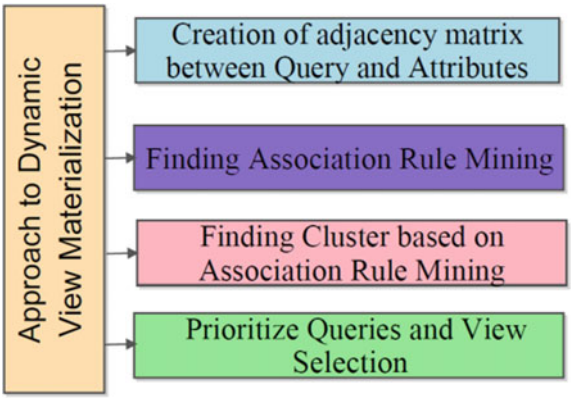

Fig. 2 Approaches to dynamic view materialization

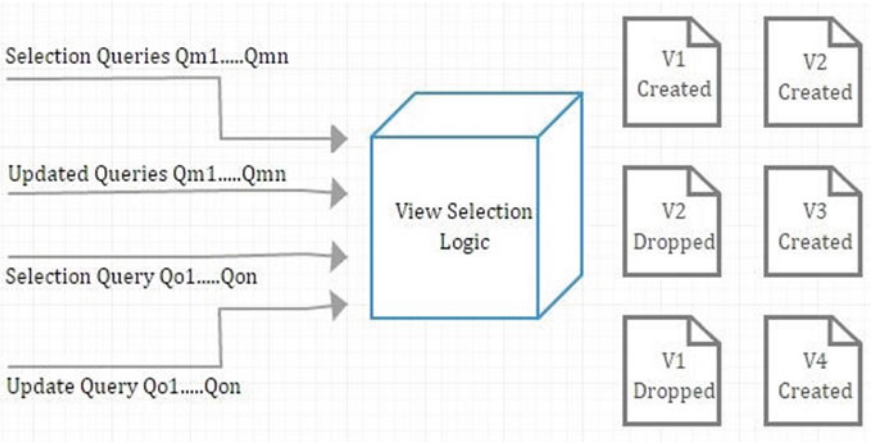

Fig. 3 View creation

3.1 Attribute Matrix Generation Table

Attribute matrix generation algorithm generates attribute matrix (M_{kj}) where k is the number of queries in the workload (Q), j is the number of attributes present in the relation/table represented by (A), and Att_{set} is the attribute set which is created from the attribute; initially, it contains φ and after the completion of this algorithm Att_{set} ends up with collection of sets of single attribute which belongs to Queries Q_j.

Table 2 *DAMS*: generated attribute matrix table

Q/*A*	At_1	At_2	At_3	At_4	At_5	At_6	At_7	At_8	At_9	At_{10}	At_{11}
Q_1	1	1	1	0	0	0	0	1	0	1	1
Q_2	0	1	1	1	1	1	1	0	0	0	0
Q_3	1	0	1	0	0	0	0	1	0	1	1
Q_4	0	0	1	1	1	1	0	0	1	0	0
Q_5	1	0	0	1	0	0	1	1	1	1	1
Q_6	0	1	1	0	0	0	1	0	1	0	0

Algorithm 1 Algorithm for Attribute Matrix

Input: Queries(Q) {1.....k}
Attribute(A) {1.....j}
$Att_{set} = \phi$
Output: Attribute Matrix M_{kj}

```
for Q 1 to k do
    for A 1 to j do
        if A[j] ∈ Q_k then
            M_kj ← 1
            Att_set = Att_set ∪ A[j]
        else
            M_kj ← 0
        end if
    end for
end for
```

That is, $Att_{set} = \{\{At_1\}\{At_2\}\{At_3\}\{At_4\}\ldots\{At_j\}\}$. Based on values of M_{kj} generated by algorithm 1, a dimension table is presented in Table 2. All possible attributes present in the query workload $\{Q_1, Q_2, Q_3, Q_4, Q_5, Q_6\}$ are depicted in the column, and individual row represents the queries. In cell, a value '1' indicates the presence and '0' indicates the absence of attribute in the specific query. Using this attribute matrix, we can co-relate the set of queries in the workload and identify the clusters.

3.2 Mapping Association Rules to Find Clusters

The main issue is to find the view which is associated with maximum number of queries, i.e., can provide result to major portion of the workload.

Consider the queries:

```
SELECT SUM(salary) from EmployeeDB where Branch_office='Mumbai'
and Joining_year=2019 GROUP BY department.
```

In the above query, we first identify the rows with office at 'Mumbai' and year 2020 and then take average of salary using aggregate function and then arranging them on the basis of department, the same query can also be considered if we first group the data on the basis of department with branch Mumbai and year 2019 and

then apply aggregate function. So, we need to find the result faster. This is depicted in Fig. 4.

Consider the set of items ITEM(I) = Bread, Milk, Butter which are frequent part of your transaction, and same is shown in Table 3, where presence and absence of attribute are shown with numeric values 1 and 0, respectively, and from a collection of transaction we can identify the closely related attribute which occurs in transaction more frequently. In online transaction, this rule will help the customers to take decision on the current trends irrespective of the requirement sometimes, and in this

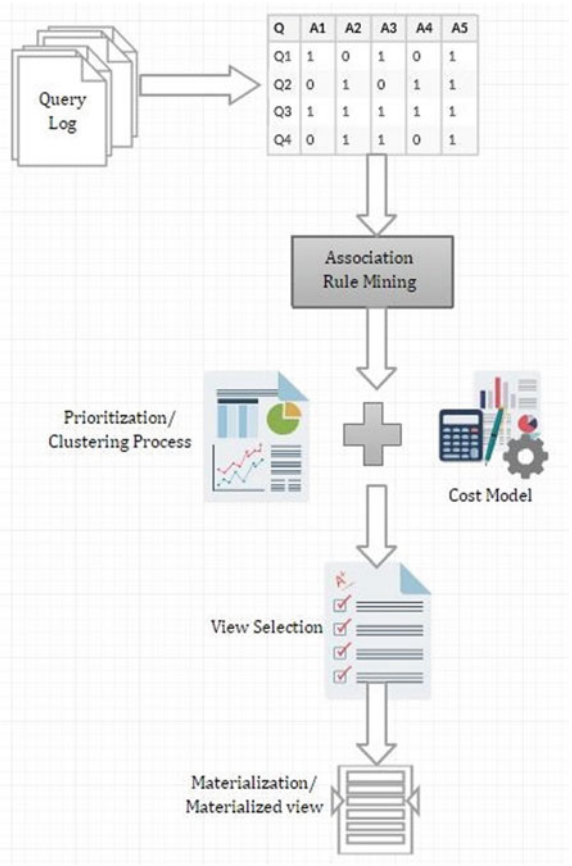

Fig. 4 Proposed architectural diagram for view materialization

Table 3 Mapping of generated associated values to rule sets

Transaction ID	Bread	Milk	Butter
1	0	1	0
2	1	1	1
3	1	1	0
4	0	0	1

way, you get suggestion on e-commerce and social media platform [4]. The row in the attribute matrix represents in the following manner.

$$\text{Q1} = 10100, \text{Q2} = 00011$$

We can understand the presence and absence of attribute in the bit vector. Now to find the co-relation between multiple queries, we can make use of bit-wise logical operator.

3.3 Clustering and Prioritizing Queries

Based on bit vector supporting a query set, we identify the attribute, create a cluster of it, then prioritize them based on the frequency in the transaction and create a cluster based on the decision support system for view selection. Consider Table 4 storing the bit value in the matrix for each query and attribute.

According to Agrawal et al. [1], ARM problem is defined as: Let $A = \{\text{At}_1, \text{At}_2, \text{At}_3, \text{At}_4, \ldots\}$ be a set of attributes from the query workload which belongs to a transaction we call it items. Let $Q = \{Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, \ldots\}$ be a set of queries in the workload. To understand this, consider each query with a unique identification which contains subset from the item set I, and A rule is defined as $A \rightarrow B$ where A, $B \subseteq S$ and $A \cap B = \text{NULL}$ [2].

To identify the interesting co-relation from the set of transactions, constraint can be applied. The key constraint to draw any conclusion in this system is use of minimum support and confidence as a threshold.

Support: The support supp($A \Rightarrow B$) of an items $A \cup B$ is defined as the frequency of data A and B comes together in a transaction data set.

$$\text{Support}(A \Rightarrow B) = \frac{n(A \cup B)}{N}$$

Table 4 To generate clusters and prioritize query sets

Query	Freq	At_1	At_2	At_3	At_4	At_5
Q_1	1	1	0	1	0	0
Q_2	1	0	1	1	0	0
Q_3	1	1	0	0	1	0
Q_4	1	0	1	0	1	1
Q_5	1	1	0	1	0	0
Q_6	1	0	1	1	1	1
Q_7	1	1	0	0	1	1
Q_8	1	0	1	1	1	0

where N is count of total transaction. If in a total of 1000 transaction the frequency of A and B comes together 150, then Support $(A \cup B) = 150/1000 = 0.15$ or 15%.

Confidence: The confidence is defined as association between item A and B, i.e., how likely B is selected if A is considered.

$$\text{Confidence}(A \Rightarrow B) = \frac{\text{Support}(A \Rightarrow B)}{\text{Support}(A)}$$

For example, if the value of confidence is $0.15/0.15 = 1$, i.e., (100%) it means that for 100% of the transactions, if attribute B is present, then A is also present, so we consider both the attributes in the same cluster. The reason behind the association rule is to create cluster based on the priority of the view. To prioritize, we have chosen A-priori algorithms, because of its efficient approach to categorize dataset items [2]. To understand the concept, consider hypothetical data along with the frequency in the query workload.

Consider the item set of one and two attributes initially.

1. Attribute set: $\{\{At_1\}, \{At_2\}, \{At_3\}, \{At_4\}, \{At_5\}\}$
2. Attribute set $\{\{At_1, At_2\}, \{At_1, At_3\}, \{At_3, At_4\}, \ldots$ so on$\}$
3. Attribute set: contains combination of 3 attributes together and so on, so this attribute set will depend on the number of attributes present in the table.

So, we first find the frequent set with 1 attribute alone in the database which we call L_1 and likewise we calculate till L_k.

L_k = attribute set of K frequent attribute.

Step 1: find the frequent set with 1 attribute in the database.

$L_k = \{\{At_1\}, \{At_2\}, \{At_3\}, \{At_4\}, \{At_5\}\}$

Next step to identify the frequent set with 1 attribute by observing L_1.

$C_2 = \{\{At_1, At_2\}, \{At_1, At_3\}, \{At_1, At_4\}, \{At_1, At_5\}, \{At_2, At_3\}, \{At_2, At_4\}, \{At_2, At_5\}, \{At_3, At_4\}, \{At_3, At_5\}, \{At_4, At_5\}\}$

Now scan the attribute table for $\{At_i, At_j\}$ existing in a Q_i in the attribute table, where $i, j \in \{At_1, At_2, At_3, At_4, At_5\}$ and $Q_i \in$ query workload.

$L_2 = \{\{At_1, At_2\}, \{At_1, At_3\}, \{At_1, At_4\}, \{At_1, At_5\}, \{At_2, At_3\}, \{At_2, At_4\}, \{At_2, At_5\}, \{At_3, At_4\}, \{At_3, At_5\}, \{At_4, At_5\}\}$

On passing minimum support and confidence on L_2, we generate C_3.

C_3 is generated by combining all possible sets of L_2.

$C_3 = \{\{At_1, At_2, At_3\}, \{At_1, At_2, At_4\}, \{At_1, At_2, At_5\}, \ldots$ and so on$\}$

Using C_3, we will get L_3 and let us assume only two sets finally qualify the minimum threshold.

$L_3 = \{\{At_2, At_3, At_4\}\ \{At_2, At_4, At_5\}\}$

$C_4 = \varphi$ and so we stop here for clustering of attribute because no further possible attribute set is capable of qualifying threshold value of set L_k, and we also calculate the support and confidence of all attribute set.

For example, attribute set $\{At_2, At_4, At_5\}$

Confidence($\{At_2, At_4\} \Rightarrow \{At_5\}$)

Confidence($\{At_2, At_5\} \Rightarrow \{At_4\}$)

Confidence($\{At_4, At_5\} \Rightarrow \{At_2\}$)

Confidence($\{At_2, At_4\} \Rightarrow \{At_5\}$)

Confidence($\{At_5\} \Rightarrow \{At_2, At_4\}$)

Confidence($\{At_2\} \Rightarrow \{At_4, At_5\}$)

Confidence($\{At_4\} \Rightarrow \{At_2, At_5\}$)

Support($\{At_2, At_4\} \Rightarrow \{At_5\}$)

and so on, and we can calculate the other support rule for this above case.

This result can help us find more relevant and co-related attribute set which helps prioritize the strong associations and how strongly they are connected together; this helps us to keep them in a same cluster and can be resolved in a same view.

Algorithm 2 Algorithm for Attribute selection using candidate set

Input: Queries(Q) {1.....k}
Attribute(A) {1.....j}
Att_{set}
C_m(candidate set)={}
L_m(minimal set)=ϕ
Output: Candidate set C of attribute

```
while Q_m ≠ φ do
    for Att_set ← 1 to j do
        for Q ← 1 to k do
            if ∀ Att_set ∈ A, Att_set ∈ Q_k then
                n(Att_set) ← n(Att_set) + 1
            end if
        end for
        C_m ← map(n(Att_set),j)
    end for
    for ∀Att_set ← 1 to j do
        if C_m[j] ≥ min_count then
            L_m[j] ← C_m[j]
        end if
    end for
    C_{m+1} ← L_m X L_m
    j ← length(C_{m+1})
end while
```

3.4 View Selection by Attribute Selection Using Candidate Sets

Attribute matrix generation algorithm is presented that generates possible candidate views for materialization. The following Algorithm 2 is used for finding the candidate set of attributes which are frequent and need to be materialized for efficient access and storage, where k is the number of queries represented by (Q), j is the number of attributes present in the relation/table represented by (A), and Att_{set} is the attribute set which was calculated in above Algorithm 1 and will be updated in each iteration using Cartesian product of previous sets which is combination pair of all the previous set element, i.e., state after 1st iteration $\text{Att}_{\text{set}} = \{\{\text{At}_1, \text{At}_2\}, \{\text{At}_1, \text{At}_3\}, \{\text{At}_1, \text{At}_4\}, \{\text{At}_1, \text{At}_5\}, \{\text{At}_2, \text{At}_3\}, \{\text{At}_2, \text{At}_4\}, \{\text{At}_2, \text{At}_5\}, \{\text{At}_3, \text{At}_4\}, \{\text{At}_3, \text{At}_5\}, \{\text{At}_4, \text{At}_5\}\}$ state after 2nd iteration $\text{Att}_{\text{set}} = \{\{\text{At}_1, \text{At}_2, \text{At}_3\}, \{\text{At}_1, \text{At}_2, \text{At}_4\}, \{\text{At}_1, \text{At}_2, \text{At}_5\}, \ldots$ and so on$\}$. C_m is the candidate set which counts the occurrence of the attribute in the query, and those which qualify minimum support count in the occurrence of queries are shifted in least count set L_m where m denotes the number of set required to find out the minimum attribute which is considered for materialization process.

Now we materialized only those view which are capable of providing answers to multiple queries, and will reduce the candidate set. Query order permutation and frequency of past queries are also used to reduce the overall query processing.

4 Research Challenges and Future

In VM, refresh needs to be performed constantly on the database object whenever an update occurs. These updates need to be propagated at real time to simultaneous users. The challenge lies in identifying the updates that are consistent among all users. The research challenges are now presented as follows

- **Partial versus Complete Refresh of View**: The refreshes are sometimes partial, where the updates are applied as they happen and the view is presented. This increases the efficiency of query updates but may introduce inconsistency among parallel users operating simultaneously. The refreshes are sometimes complete, where all updates first occur on the VM and then the view is presented to users. This allows consistency among all users for a common view but increases the latency of the query. Fine-tuning an appropriate balance is a difficult task.
- **Cost of Materialization**: Another challenge is establishing a trade-off among limited memory in constrained environments against the query execution time. Increasing memory usage reduces the execution time of fetching queries, which is not feasible in low-powered environments.
- **Attribute Selection**: Another challenge is identifying the proper sets of frequent attributes during VM. Frequent attributes may be pre-fetched so that latency reduces. The challenge is to determine the estimated range value of attributes that needs to be pre-fetched.

As rule-based queries have casual dependencies, the cost of querying view increases. To address the same, the authors as part of future work would propose machine-learning-based approaches on graph-based models to classify a set of candidate views to be materialized. This would reduce the overall query and maintenance cost of updating views. To address the same, queries are clustered as graph datasets. A priority algorithm for the selection of materialized views would be proposed. This reduces the overall storage and query processing cost.

5 Conclusion and Future Work

Optimal VS requires materialization in order to minimize the overall data retrieval time. In the proposed approach *DAMS*, based on assigned workloads, similarity and dissimilarity of features are grouped into cluster sets based on confidence values. To define the candidate view, we use clusters of related queries which further solve multiple queries. If candidate views are in number, it will take more time to identify which candidate is merged and at the same time ensure the capability to answer multiple queries. The concept rule mining provides much stronger associated attribute set. It also helps in the clustering of related attribute which is frequent in queries in a database for a given amount of time period.

References

1. Agrawal, R., Imieliundefinedski, T., Swami, A.: Mining association rules between sets of items in large databases. SIGMOD Rec. **22**(2), 207–216 (1993)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proceedings of 20th International Conference on Very Large Data Bases VLDB1215 (08 2000)
3. Anter, S., Zellou, A., Idri, A.: personalization of a hybrid integration system: creation of views to materialize based on the distribution of user queries. In: 2012 IEEE International Conference on Complex Systems (ICCS), pp. 1–7. IEEE (2012)
4. Aouiche, K., Jouve, P.E., Darmont, J.: Clustering-based materialized view selection in data warehouses. In: Manolopoulos, Y., Pokorný, J., Sellis, T.K. (eds.) Advances in Databases and Information Systems, pp. 81–95. Springer, Berlin (2006)
5. Arab, M., Sohrabi, M.K.: Proposing a new clustering method to detect phishing websites. Turkish J. Electr. Eng. Comput. Sci. **25**(6), 4757–4767 (2017)
6. Azgomi, H., Sohrabi, M.K.: A game theory-based framework for materialized view selection in data warehouses. Eng. Appl. Artif. Intell. **71**, 125–137 (2018)
7. Bhattacharya, P., Tanwar, S., Bodke, U., Tyagi, S., Kumar, N.: Bindaas: blockchain-based deep-learning as-a-service in healthcare 4.0 applications. IEEE Trans. Netw. Sci. Eng. 1–1 (2019). https://doi.org/10.1109/TNSE.2019.2961932
8. Bhattacharya, P., Tanwar, S., Shah, R., Ladha, A.: Mobile edge computing-enabled blockchain framework—a survey. In: Singh, P.K., Kar, A.K., Singh, Y., Kolekar, M.H., Tan-war, S. (eds.) Proceedings of ICRIC 2019. pp. 797–809. Springer International Publishing, Cham (2020)
9. Bhattacharya, P., Tiwari, A.K., Srivastava, R.: Dual buffers optical based packet switch incorporating arrayed waveguide gratings. J. Eng. Res. **7**, 1–15 (2019)

10. Bodkhe, U., Bhattacharya, P., Tanwar, S., Tyagi, S., Kumar, N., Obaidat, M.S.: Blohost: blockchain enabled smart tourism and hospitality management. In: 2019 International Conference on Computer, Information and Telecommunication Systems (CITS). pp. 1–5 (Aug2019)
11. Cavero Barca, J.M., Sánchez, B.V., García de Marina, P.C.: Evaluation of an implementation of cross-row constraints using materialized views. ACM SIGMOD Record **48**(3), 23–28 (2019)
12. Ehsan, H., Sharaf, M.A.: Materialized view selection for aggregate view recommendation. In: Chang, L., Gan, J., Cao, X. (eds.) Databases Theory and Applications, pp. 104–118. Springer International Publishing, Cham (2019)
13. Gosain, A., Madaan, H.: Efficient approach for view materialization in a data warehouse by prioritizing data cubes. IET Softw. **12**(6), 498–506 (2018)
14. Gosain, A., Madaan, H.: Query Prioritization for View Selection, pp. 403–410 (07 2018). https://doi.org/10.1007/978-981-10-3373-540
15. Hemmatian, F., Sohrabi, M.K.: A survey on classification techniques for opinion mining and sentiment analysis. Artif. Intell. Rev. 1–51 (2017)
16. Jindal, A., Karanasos, K., Rao, S., Patel, H.: Selecting subexpressions to materialize at data-center scale. Proc. VLDB Endow. **11**(7), 800–812 (2018)
17. Kabra, N., Bhattacharya, P., Tanwar, S., Tyagi, S.: Mudrachain: blockchain-based frame-work for automated cheque clearance in financial institutions. Futur. Gen. Comput. Syst. **102**, 574–587 (2020)
18. Kumar, A., Kumar, T.V.: Materialized view selection using discrete genetic operators-based particle swarm optimization. In: 2017 International Conference on Inventive Systems and Control (ICISC), pp. 1–5. IEEE (2017)
19. Masunaga, Y.: An intention-based approach to the updatability of views in relational databases. In: Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication, p. 13. ACM (2017)
20. Phan, T., Li, W.: Dynamic materialization of query views for data warehouse workloads. In: 2008 IEEE 24th International Conference on Data Engineering, pp. 436–445 (April 2008)
21. Rashid, A.N.M.B., Islam, M.S.: An incremental view materialization approach in ordbms. In: 2010 International Conference on Recent Trends in Information, Telecommunication and Computing, pp. 105–109 (March 2010)
22. Sohrabi, M.K., Akbari, S.: A comprehensive study on the effects of using data mining techniques to predict tie strength. Comput. Hum. Behav. **60**, 534–541 (2016)
23. Sohrabi, M.K., Azgomi, H.: Evolutionary game theory approach to materialized view selection in data warehouses. Knowl.-Based Syst. **163**, 558–571 (2019)
24. Sohrabi, M.K., Hasannejad, M.H.: Association rule mining using new fp-linked list algorithm (2016)
25. Srivastava, A., Bhattacharya, Singh, A., Mathur, A., Prakash, O., Pradhan, R.: A distributed credit transfer educational framework based on blockchain. In: 2018 Second International Conference on Advances in Computing, Control and Communication Technology (IAC3T), Allahabad, India, pp. 54–59. IEEE (2018)
26. Xu, W., Theodoratos, D., Zuzarte, C., Wu, X., Oria, V.: A dynamic view materialization scheme for sequences of query and update statements. In: Song, I.Y., Eder, J., Nguyen, T.M. (eds.) Data Warehousing and Knowledge Discovery, pp. 55–65. Springer, Berlin (2007)
27. Ye, Y., Liu, X., Yin, J.: Multi-view clustering with noisy views. In: Proceedings of the 20182nd International Conference on Computer Science and Artificial Intelligence, pp. 339–344 (2018)
28. Zlamaniec, T., Chao, K.M., Godwin, N., Shah, N., Farmer, R.: A framework for workload-aware views materialisation of semantic databases. In: 2015 IEEE 12th International Conference on e-Business Engineering, pp. 15–22. IEEE (2015)