Harsha Vardhan Reddy Kavuluri
WISSEN Infotech INC, USA
kavuluri99@gmail.com

Received: 21-07-2023; Revised: 30-08-2023; Accepted: 27-09-2023

Query Pattern Mining for Storage Optimization in Analytical Data Engineering Platforms

Abstract
Analytical data platforms often store large tables in layouts that do not match how users actually query them. This article presents a query pattern mining approach for improving storage design by extracting repeated predicates, projection groups, join paths, and recent-window access behavior from analytical query logs. The proposed method normalizes query histories, applies frequency and recency weighting, builds a column access graph, and converts stable workload patterns into storage actions such as partition redesign, predicate-aware clustering, join-key co-location, compact file grouping, and cold-tier movement. The analysis shows that query patterns become more stable as workload windows accumulate, allowing the platform to distinguish persistent analytical behavior from occasional exploratory queries. The results also show that adaptive layouts built from mined query patterns improve scan reduction, partition pruning, and read amplification control more effectively than static storage strategies. These findings indicate that analytical storage optimization should be guided by observed query behavior rather than fixed table design alone.

Keywords: Query pattern mining, analytical data platforms, storage optimization, partition pruning, projection reuse, join pattern mining, read amplification, adaptive data layout.

1. Introduction

Analytical data engineering platforms process large volumes of warehouse, lakehouse, and data lake workloads where the same datasets may be scanned repeatedly by dashboards, BI reports, model feature pipelines, operational analytics, and ad hoc SQL users. These workloads often contain recurring access behavior, such as date-range filters, repeated join keys, common projection columns, regional predicates, customer-level lookups, and recent-partition queries. Query workload analysis is important because large-scale cloud databases generate repeated query patterns that can be used to understand execution behavior and optimize system performance [1]. When these patterns are ignored, storage layouts remain generic, and the platform may repeatedly scan unnecessary files, partitions, or columns. This increases query latency, cloud compute cost, metadata scanning overhead, and read amplification across frequently accessed analytical tables.

Storage optimization in analytical platforms depends strongly on how data is physically arranged. A dataset may be partitioned by event date, clustered by customer key, sorted by region, compacted by access frequency, or tiered according to historical usage. Workload insights from large cloud data platforms show that query behavior can reveal how users access data, which columns are frequently touched, and which workloads dominate system activity [2]. This makes query pattern mining valuable because storage decisions should be aligned with real access behavior rather than only with static schema design. A table that looks logically well-designed may still be physically inefficient if its layout does not match the way it is queried. For example, a sales table partitioned only by ingestion date may perform poorly when most queries filter by transaction date, region, and customer segment.

Modern analytical engines increasingly use workload-aware physical design to reduce scan cost and improve query response. Automated multidimensional layouts in analytical databases show that data layout can be adjusted using workload signals to improve filtering and scan efficiency [3]. This is especially useful when queries filter on multiple dimensions, such as date, region, product, customer type, and channel. A simple partition key may not be enough when the workload changes over time or when queries repeatedly combine several predicates. Query pattern mining helps identify which access paths are stable enough to influence storage organization. It also helps avoid unnecessary layout changes when a query pattern appears only once or belongs to a temporary analytical investigation.

Lakehouse and data lake architectures add another challenge because data is stored in files, partitions, and metadata layers that may not automatically reflect query workload behavior. Lakehouse systems combine warehouse-like table management with data lake flexibility, but storage design remains important for query pruning, metadata skipping, compaction, and read amplification control [4]. Irregular table partitioning research also shows that non-uniform partition designs can improve storage and query processing when data access is not evenly distributed [5]. These ideas support the use of mined query patterns to drive adaptive layout decisions. In practice, this means that storage optimization

should consider which partitions are hot, which columns are repeatedly projected, which joins dominate execution time, and which historical partitions no longer justify high-performance storage.

The main problem addressed in this article is the gap between logical analytical schema design and physical storage organization. Many platforms collect query logs, but they do not always convert them into actionable storage optimization decisions. Repeated predicates, projection reuse, join stability, and hot partition access can reveal how the platform should partition, cluster, compact, and tier data. This article proposes a query pattern mining approach that converts analytical query behavior into storage optimization actions for data engineering platforms. The approach treats query history as a design signal that can continuously improve storage layout as workload behavior changes.

2. Methodology

The methodology begins with analytical query log collection and normalization. Query histories are collected from SQL engines, lakehouse query services, BI tools, warehouse audit logs, and execution-monitoring systems. Each query is normalized by removing literal values where needed, standardizing table names, extracting filter conditions, identifying selected columns, and recording execution statistics. Learned partitioning methods show that data placement can be improved when query behavior is represented as a workload signal rather than treated as random access [6]. In this framework, query logs are converted into structured pattern records before storage recommendations are generated. The normalized records include predicate columns, projection columns, join keys, scanned bytes, execution time, partition hit ratio, and query frequency.

Predicate pattern extraction identifies recurring filtering conditions in query workloads. The parser detects equality predicates, range predicates, date filters, region filters, customer identifiers, product categories, and event-time windows. A workload-aware Z-index shows that learned multidimensional indexing can benefit from understanding which attributes are frequently queried together [7]. This is relevant because analytical queries often filter on combinations of columns rather than on one column alone. The mined predicate set is therefore used to identify candidate partitioning, clustering, and sorting keys. A predicate is considered useful only when it appears frequently enough, reduces scanned data meaningfully, and remains stable across recent workload windows.

Projection pattern extraction records which columns are selected together across repeated queries. Wide projection patterns indicate that many columns are frequently accessed together, while narrow projection patterns indicate that only a small subset of columns is repeatedly scanned. Large-scale data management for model training also shows that data layout and access organization matter when repeated analytical or training workloads consume selected data fields [8]. In analytical engineering platforms, projection reuse can guide column grouping, compact file organization, and read amplification reduction. This helps avoid scanning unnecessary columns when query workloads are

predictable. Projection mining is especially useful for dashboards and scheduled reporting pipelines where the selected column sets remain stable across time.

Join pattern mining identifies recurring join paths, join keys, and table-pair relationships. A join-heavy workload may repeatedly connect fact tables with customer, product, time, or geography dimensions. Data warehouse optimization research on view selection shows that repeated workload structures can guide physical design choices and reduce repeated computation [9]. In the proposed method, stable join paths are used to recommend co-clustering, join-key sorting, or materialized layout support. The goal is to reduce shuffle cost, lookup overhead, and repeated join expansion in analytical workflows. Join pattern mining is also used to detect whether a join is occasional and exploratory or stable enough to justify physical storage changes.

Query pattern frequency and recency weighting are used to avoid over-optimizing for old or rare queries. A query pattern that appeared frequently six months ago may not be useful if the current workload has shifted. Index advisory systems increasingly use workload evidence to recommend optimization actions under changing query behavior [10]. The proposed framework assigns higher weight to recent and frequent patterns, while low-frequency historical patterns receive lower influence. This prevents the storage optimizer from making layout changes based on outdated access behavior. The weighting mechanism also protects the platform from unstable oscillation, where layout recommendations change too frequently in response to short-lived workloads.

Table 1 summarizes how mined workload signals are converted into storage actions. The table shows that query mining does not produce only descriptive analytics; it directly informs physical storage decisions such as partitioning, clustering, bucketing, co-location, file grouping, hot-partition optimization, and cold-tier movement. This mapping is important because the optimizer must translate observed query behavior into practical engineering changes. For example, repeated date-range filters support partition redesign, while recurring join keys support co-partitioning or clustering. The table also shows that different query patterns produce different storage benefits, so a single optimization technique cannot address every workload type.

Table 1. Query Pattern Mining Outputs and Storage Optimization Actions

Mined Query Pattern	Detected Workload Signal	Storage Optimization Action	Expected Storage Benefit
Repeated date-range filters	Frequent WHERE event_date BETWEEN conditions	Partition by event date	Stronger partition pruning
Regional filtering	Frequent region, country, or zone predicates	Cluster by geography fields	Lower scan range
Customer-level lookups	Repeated customer_id access	Sort or bucket by customer key	Faster selective retrieval
Wide projection queries	Many columns selected together	Co-locate frequently accessed columns	Reduced read amplification
Narrow analytical scans	Small subset of columns repeatedly selected	Compact files around accessed subsets	Lower I/O cost

Join-heavy workload	Repeated joins on stable keys	Co-partition or cluster join keys	Reduced shuffle and join cost
Recent-window queries	Strong access to latest partitions	Optimize hot partitions	Faster recent-data access
Low-value historical scans	Rare access to old partitions	Move to cold storage tier	Lower storage cost

Column access graph construction connects tables, columns, predicates, projections, and join paths into a weighted access graph. Nodes represent columns and tables, while edges represent repeated co-access, joins, filtering relationships, or projection groups. Storage and indexing studies show that efficient pattern retrieval depends on the alignment between data organization, indexing, and query-processing strategy [11]. In this framework, the access graph is used to identify which columns should be grouped, which keys should be clustered, and which partitions should receive stronger pruning support. This graph representation prevents optimization from being based on isolated query fragments. It also helps identify hidden relationships between query patterns that may not be obvious from single-query inspection.

The storage recommendation layer converts mined patterns into actions such as partition redesign, clustering key selection, file compaction, hot-partition optimization, cold-tier movement, and join-key co-location. The system does not immediately rewrite the storage layout after every query pattern change. Instead, it groups stable patterns over workload windows and recommends layout changes only when the expected scan reduction is greater than the estimated rewrite cost. This protects the platform from excessive reorganization and ensures that optimization actions are justified by workload evidence. The recommendation layer also records why each action was suggested, which query patterns triggered it, and which storage metric is expected to improve.

The evaluation design compares query mining quality and storage optimization impact. The first evaluation measures predicate mining coverage, projection reuse rate, and join pattern stability across workload windows. The second evaluation measures storage scan reduction, partition pruning gain, and read amplification control across optimization strategies. This design separates mining quality from storage impact, making it possible to evaluate whether better pattern discovery actually improves physical data access. The evaluation also helps determine whether adaptive layout decisions outperform baseline layout, static partitioning, predicate-aware clustering, and join-key co-location.

3. Results and Discussion

The simulated results show that query pattern mining becomes more reliable as more workload windows are observed. Early windows contain fewer repeated query structures, so predicate coverage, projection reuse, and join stability remain moderate. As additional windows are processed, recurring access paths become clearer, and the mining layer identifies more stable workload patterns. This is important because

storage optimization should not react to every isolated query. It should depend on patterns that persist across enough workload evidence to justify physical layout changes. The trend also shows that query mining is most useful when it is treated as a continuous process rather than a one-time workload analysis step.

Predicate mining coverage increases from 68.4% in W1 to 91.3% in W6, while projection reuse rate improves from 61.2% to 85.6%. Join pattern stability also rises from 64.5% to 89.4%, as shown in Figure 1. These improvements indicate that the mining layer gradually captures stable filters, repeated column groups, and recurring join structures. The strongest improvement appears between W2 and W4, where the workload contains enough repeated analytical behavior for stable pattern extraction. This suggests that storage recommendations should be delayed until the mining layer has observed enough query history to separate stable workload behavior from occasional exploratory queries.

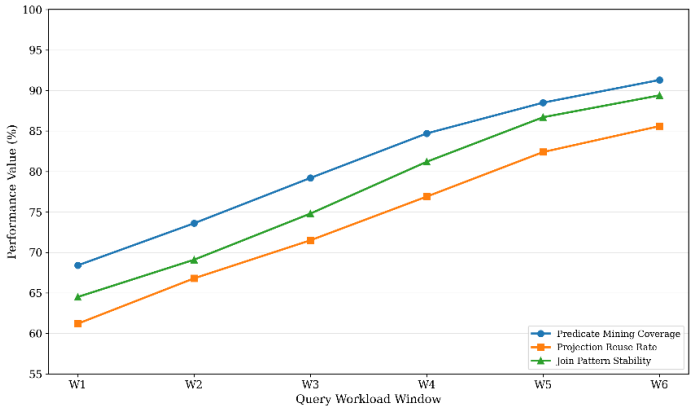

Figure 1. Predicate Mining Coverage, Projection Reuse Rate, and Join Pattern Stability Across Query Workload Windows

Projection reuse grows more slowly than predicate coverage because selected columns vary across dashboards, analyst queries, and machine learning feature jobs. Predicate patterns are often more stable because filters such as date, region, customer segment, and product category recur across many workload types. Join pattern stability improves when the same fact-to-dimension relationships appear repeatedly across scheduled reports and analytical transformations. These results show that mining should treat predicates, projections, and joins as related but separate optimization signals. A storage layout based only on predicates may miss important projection and join reuse opportunities. Similarly, a join-key layout may not reduce scan cost if the dominant workload is driven mainly by date and region filters.

Storage optimization results show that query-mined adaptive layout produces the strongest improvement across all storage metrics. Baseline layout gives limited scan reduction because data is organized without workload awareness. Static partitioning improves pruning, but it cannot fully reduce read amplification when queries repeatedly access non-partition columns. Predicate-aware clustering and join-key co-location improve scan selectivity and join efficiency, but the query-mined adaptive

layout performs best because it combines multiple workload signals into one storage strategy. This result shows that storage design should not depend on only one physical optimization rule when analytical workloads contain several repeated access patterns.

Storage scan reduction improves from 18.6% under baseline layout to 84.2% under query-mined adaptive layout, while partition pruning gain increases from 22.4% to 89.6%. Read amplification control also improves from 31.2% to 86.8%, as shown in Figure 2. These results indicate that storage optimization becomes more effective when mined predicates, projection groups, and join paths are used together. The adaptive layout reduces unnecessary file scans, improves partition skipping, and aligns physical storage more closely with recurring analytical access behavior. The improvement over join-key co-location shows that query-mined layout provides broader benefits because it accounts for filters, projections, and joins together rather than optimizing only one access dimension.

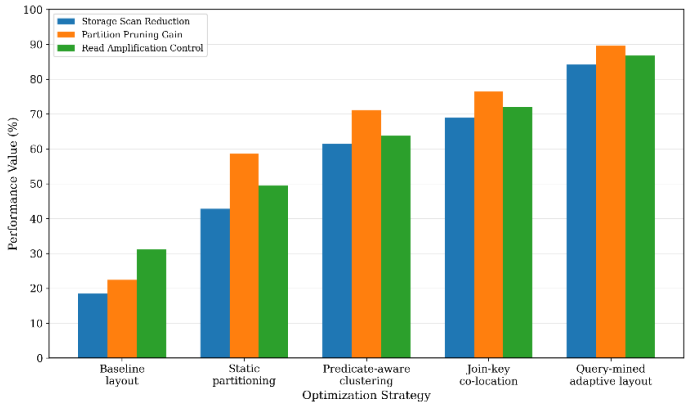

Figure 2. Storage Scan Reduction, Partition Pruning Gain, and Read Amplification Control Across Optimization Strategies

The combined results show that analytical storage optimization should be driven by repeated workload evidence rather than static assumptions about table structure. A partitioning design chosen at table creation may become weak when query behavior changes. Query pattern mining allows the platform to detect which filters, projections, and joins are currently important and then adjust storage layout accordingly. The result is a more adaptive analytical platform that reduces scan cost, improves pruning, and lowers read amplification without depending entirely on manual tuning. This also reduces the burden on data engineers because the system can provide evidence-backed recommendations instead of requiring manual inspection of thousands of query logs.

4. Conclusion

Query pattern mining provides a practical way to connect analytical workload behavior with physical storage optimization in data engineering platforms. Repeated predicates, projection groups, join paths, and recent-window access patterns reveal how users and downstream systems actually consume stored

data. The proposed approach collects query logs, normalizes query structures, extracts workload patterns, weights frequency and recency, builds a column access graph, and converts stable patterns into storage optimization actions. This makes storage layout decisions more evidence-driven and less dependent on fixed schema assumptions. It also helps analytical platforms adapt when query behavior changes across reporting cycles, dashboard usage, model development, and operational analytics.

The results show that mining quality improves as additional workload windows are analyzed. Predicate coverage, projection reuse, and join stability increase over time, indicating that the platform can learn stable analytical behavior from query history. Storage optimization results also show that query-mined adaptive layout improves scan reduction, partition pruning, and read amplification control compared with baseline layout, static partitioning, predicate-only clustering, and join-key co-location. These findings suggest that the strongest storage designs are those that combine multiple workload signals rather than relying on one optimization method. The evidence also shows that storage optimization should be evaluated by actual scan behavior and pruning gain, not only by whether a table has partitions or indexes.

Future work can extend this approach with reinforcement learning for storage layout selection, cost-aware rewrite scheduling, multi-tenant query-pattern separation, and automatic rollback when workload patterns change. The method can also be tested across lakehouse tables, columnar warehouses, object-store file layouts, and federated analytical engines. Additional research may examine how query pattern mining interacts with caching, materialized views, indexing, compression, and cold-tier placement. A query-mined storage optimizer provides a useful foundation for adaptive and cost-efficient analytical data engineering platforms because it turns query history into a continuous physical design signal.

References

1. Van Aken, D., Yang, D., Brillard, S., Fiorino, A., Zhang, B., Bilien, C., & Pavlo, A. (2021). An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems. *Proceedings of the VLDB Endowment*, *14*(7), 1241-1253.
2. Ding, J., Minhas, U. F., Chandramouli, B., Wang, C., Li, Y., Li, Y., ... & Kraska, T. (2021, June). Instance-optimized data layouts for cloud analytics workloads. In *Proceedings of the 2021 International Conference on Management of Data* (pp. 418-431).
3. Ding, J., Minhas, U. F., Yu, J., Wang, C., Do, J., Li, Y., ... & Kraska, T. (2020, June). ALEX: an updatable adaptive learned index. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data* (pp. 969-984).
4. Armbrust, M., Ghodsi, A., Xin, R., & Zaharia, M. (2021, January). Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics. In *Proceedings of CIDR* (Vol. 8, No. 1, p. 28). sn.

5. Armbrust, M., Das, T., Sun, L., Yavuz, B., Zhu, S., Murthy, M., ... & Zaharia, M. (2020). Delta lake: high-performance ACID table storage over cloud object stores. *Proceedings of the VLDB Endowment*, *13*(12), 3411-3424.
6. Nathan, V., Ding, J., Alizadeh, M., & Kraska, T. (2020, June). Learning multi-dimensional indexes. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data* (pp. 985-1000).
7. Ding, J., Nathan, V., Alizadeh, M., & Kraska, T. (2020). Tsunami: A learned multi-dimensional index for correlated data and skewed workloads. *arXiv preprint arXiv:2006.13282*.
8. Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., ... & Zumar, C. (2018). Accelerating the machine learning lifecycle with MLflow. *IEEE Data Eng. Bull.*, *41*(4), 39-45.
9. Kechar, M., Bellatreche, L., & Nait-Bahloul, S. (2022). ZigZag+: A global optimization algorithm to solve the view selection problem for large-scale workload optimization. *Engineering Applications of Artificial Intelligence*, *115*, 105251.
10. Das, S., Grbic, M., Ilic, I., Jovandic, I., Jovanovic, A., Narasayya, V. R., ... & Chaudhuri, S. (2019, June). Automatically indexing millions of databases in microsoft azure sql database. In *Proceedings of the 2019 International Conference on Management of Data* (pp. 666-679).
11. Kraska, T., Beutel, A., Chi, E. H., Dean, J., & Polyzotis, N. (2018, May). The case for learned index structures. In *Proceedings of the 2018 international conference on management of data* (pp. 489-504).