Multi-Query Optimization for the Novel Analyze Operator

Marios Iakovidis[1,*,†], Panos Vassiliadis[1,†]

[1]Univ. Ioannina, Ioannina, Greece

Abstract

In their hunt for highlights, i.e., interesting patterns in the data, data analysts have to issue *groups* of related queries and *manually* combine their results. To the extent that the analyst's goals are based on an *intention* on what to discover (e.g., contrast a query result to peer ones, verify a pattern to a broader range of data in the data space, etc), the integration of *intentional* query operators in analytical engines can enhance the efficiency of these analytical tasks. In this paper, we introduce, with well-defined semantics, the *ANALYZE operator*, a novel cube querying intentional operator that provides a 360^o view of data. We define the semantics of an ANALYZE query as a tuple of five internal, facilitator cube queries, that (a) report on the specifics of a particular subset of the data space, which is part of the query specification, and to which we refer as the *original query*, (b) contrast the result with results from peer-subspaces, or *sibling queries*, and (c) explore the data space in lower levels of granularity via *drill-down queries*. We introduce formal query semantics for the operator and we theoretically prove that we can obtain the exact same result by merging the facilitator cube queries into a smaller number of queries. This effectively introduces a *multi-query optimization (MQO)* strategy for executing an ANALYZE query. We propose three alternative algorithms, (a) a simple execution without optimizations (*Min-MQO*), (b) a total merging of all the facilitator queries to a single one (*Max-MQO*), and (c) an intermediate strategy, *Mid-MQO*, that merges only a subset of the facilitator queries. Our experimentation demonstrates that Mid-MQO achieves consistently strong performance across several contexts, Min-MQO always follows it, and Max-MQO excels for queries where the siblings are sizable and significantly overlap.

Keywords

Data analytics, Intentional Operators, Query processing, Multiple Query Optimization

1. Introduction

Routinely, data analysts find themselves issuing *groups* of related queries in an attempt to uncover hidden gems in the data. Such significant findings, or *highlights*, are practically verifications of the existence of interesting properties (e.g., a trend, a peak, an outlier, etc) in a subset of the data that is of interest. Automating the extraction of highlights is, therefore, an important task in accelerating the work of data analysts, and allowing for more extensive and complete exploration of the data space.

Related work on the subject of *Automated Highlight Extraction* provides tools for multidimensional data exploration to discover interesting data patterns [1, 2, 3, 4, 5]. The primary focus of these tools is to explore a data space, find data patterns, and measure how interesting they are using interestingness metrics. However, the research landscape is characterized by (a) the absence of a widely accepted formal model that encompasses the proposed tasks in a coherent framework (the Intentional Model of [6, 7] is a first attempt), (b) at large, the omission of the opportunity to utilize multidimensional hierarchical data spaces (in the Business Intelligence sense), by focusing almost solely on relational data, and (c) to the best of our knowledge, the absence of an operator that provides an all-encompassing view of the data of interest to an analyst in a single invocation.

In this paper, we propose the *ANALYZE operator*, a novel cube query operator that formalizes the intention of providing a 360^o view of the data to the data analyst, via a single invocation. Being an algebraic operator, ANALYZE comes with (a) a well-defined multidimensional, hierarchical underlying data model with data cubes and dimensions as its basic elements [8], (b) well defined semantics on

DOLAP 2026: 28th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data, co-located with EDBT/ICDT 2026, March 24, 2026, Tampere, Finland

*Corresponding author.

†These authors contributed equally.

miakovidis@cs.uoi.gr (M. Iakovidis); pvassil@cs.uoi.gr (P. Vassiliadis)

0009-0006-9061-3450 (M. Iakovidis); 0000-0003-0085-6776 (P. Vassiliadis)

the expected results, and also, in our case, (c) optimization strategies for its efficient execution. The simplicity of a hierarchical multidimensional model facilitates a simple query operator that exploits the model's ability to compute sibling, ancestor, and descendant values at different levels of coarseness smoothly. The semantics of an ANALYZE query include five internal, *facilitator* cube queries. First, the operator has to deal with the information requested for a very specific subset of the data space that is obtained via (a) a set of filters and (b) a pair of groupers that are used in a GROUP-BY fashion to aggregate a measure. We call this the *original query* of the operator as it targets a subset of the data space of interest. Second, we want to contrast the result with similar, peer results. To this end, we use the combination of grouper and filter conditions to obtain *sibling* subspaces of the data that query subspaces sharing the same ancestor values with the filters of the original query. Finally, we explore the data space in lower levels of granularity via what we call *drill-down queries* (very much in the traditional OLAP sense).

Apart from introducing formal query semantics for the operator, however, we have been able to theoretically prove that we can obtain the exact same result by merging the facilitator cube queries into a smaller number of merged queries that exploit cube usability results to reduce redundant computation. This theoretical result effectively allows us to introduce a *multi-query optimization (MQO)* strategy that merges several collaborator queries into one or more merged queries, in an attempt to speed up the execution of an ANALYZE query. In fact, we have devised several strategies of Multi-Query Optimization for merging the underlying facilitator queries into one. In the simplest case, no optimization is performed and the five facilitator queries are executed independently: we refer to this as *Min-MQO*, as it introduces the least degree of query merging. Second, we follow the theoretical result and merge all the facilitator queries into a single one, a strategy that we call *Max-MQO*, as it implements the original theoretical result of merging everything. Finally, based on our original experimental observations that identified cases of low performance for the aforementioned strategies, we also introduce an intermediate strategy, *Mid-MQO*, that strikes a balance between the two extremes and merges only a subset of the facilitator queries. In all merging strategies, once the results of the merged queries are obtained, they are post-processed, such that the exact facilitator queries are populated correctly.

We have extensively evaluated the performance of each MQO strategy via various query workloads on multiple datasets, to assess both the effect of data size and query selectivity to the execution cost as well as the optimal strategy. Our experiments demonstrate that *Mid-MQO* is the most efficient algorithm that scales smoothly with data size and selectivity and typically achieves the best performance. Although in many cases *Max-MQO* has the worst performance, when the sibling queries are sizable and significantly overlap, *Max-MQO* is the fastest algorithm; we can predict via a decision tree when this is the case. Finally, *Min-MQO* is never the optimal strategy, although it follows *Mid-MQO* fairly closely.

Roadmap. Section 2 reviews related work in exploratory data analysis and pattern discovery. Section 3 introduces the formal modeling background. Section 4 presents the design and semantics of the ANALYZE operator. Section 5 develops our multi-query optimization strategies. Section 6 reports our experimental evaluation. Section 7 concludes with a discussion of the findings and future directions. A long version of this paper, with more results, details and proofs, appears at [9].

2. Related Work

Automated EDA. Exploratory Data Analysis (EDA) [10] allows data analysts to interact with dataset management tools to gain highlights. The goal of EDA is the production of highlights, in order to find interesting, surprising and important facts of a data subspace (likely a query result) and present them using data narration methods [1, 2, 11, 12, 13, 14, 15, 16, 3, 5, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26].

Intentional Model. The Intentional Model [6, 7] envisions Business Intelligence tools, where users will apply intentional operators over data, to simplify the querying step of data analysis operations. The user expresses high-level requirements, like 'analyze', 'assess', 'predict', that have to be addressed via auxiliary queries, ML models, highlights, and data stories. Although the general framework of the intentional model is specified in [6, 7], the exact implementation of the operators is an open problem

(to which we currently contribute with respect to the ANALYZE operator). [27, 28] explore the ASSESS operator for contrasting query results to specified 'benchmarks' of expected performance. [29] presents the DESCRIBE operator, which generates cubes annotated with model components (e.g. clusters, outliers). [30] suggests an EXPLAIN operator, which uses statistical models to explain why a measure takes certain values in relation to other measures. The most similar work to the current one, and the root of the Intentional Model, is the Cinecubes method [31, 32], which produces a data story as a PowerPoint presentation with the results of auxiliary drill-down and sibling queries.

Multi-Query Optimization (MQO).[33] establishes the foundational evidence that processing multiple queries together yields considerable cost reductions compared to independent query execution. [34] provides one of the first formal treatments of MQO, as the task of generating an optimal execution strategy for a set of concurrent queries by identifying and exploiting common subexpressions, shared join paths, and reusable intermediate results. [35] discusses a comprehensive experimental validation for Volcano optimizers. [36, 37] provide extra rewriting techniques. [38] provides approximate optimization techniques.

Comparison to the State of the Art. The main contribution of this work is the formal introduction of a novel operator along with its optimization techniques. Although conceptually related, our work here does not pertain to the core of the area of multi-query optimization: our MQO algorithms do not explore alternative query execution plans, but rather exploit theoretical guarantees of correctness. Compared to the most similar line of work [31, 32], we change the query semantics to address efficiency issues (thus we face a new problem), provide formal semantics of an operator and, most importantly, we introduce multi-query optimization strategies for the speed up of query processing. Compared to the rest of the corpus of related literature, to the best of our knowledge, there is no other work with multi-query optimization strategies for a formal, single operator that provides a 360^o view of the data with opportunities for optimization exactly due to the integration of various queries.

3. Preliminaries & Formal Background

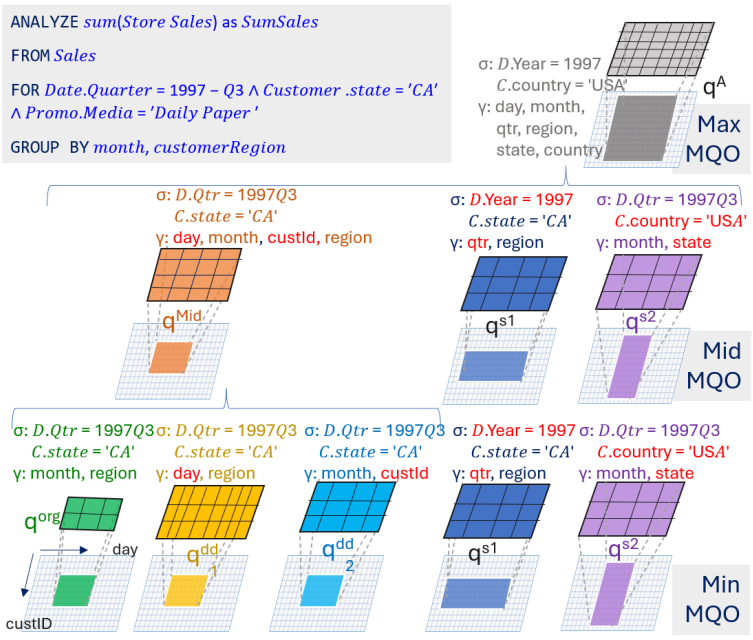


Figure 1: Facilitator queries per algorithm

In our deliberations, we assume the formal model of [8] (also used in [39]) for the definition of the multidimensional space, cubes, and cube queries.

3.1. Multidimensional Space

Multidimensional space. Data are defined in the context of a multidimensional space. The multidimensional space includes a finite set of dimensions.

A *dimension level* L includes a name and a finite set of values, $dom(L)$, as its domain. Following the traditional OLAP terminology, the values that belong to the domains of the levels are called *dimension members*, or simply *members* (e.g. the values Paris, Rome, Athens are members of the domain of level *City*, and, subsequently, of dimension

Geography). A *dimension* is a non-strict partial order of a finite set of *levels*, obligatorily including (a) a most detailed level at the lowest possible level of coarseness and (b) an upper bound, which is called *ALL*, with a single value 'All'. To ease notation, unless explicitly mentioned otherwise, in the sequel, we will assume total orders of levels, which means that there is a linear order of the levels starting from the lower level and ending at ALL with a linear chain of precedence.

We can map the members at a lower level of coarseness to values at a higher level of coarseness via an *ancestor function* $anc_{L^l}^{L^h}()$. The inverse of an ancestor function is not a function, but a mapping of a high level value to a set of *descendant values* at a lower level of coarseness and is denoted via the notation $desc_{L^h}^{L^l}()$. For example $Europe = anc_{City}^{Continent}(Athens)$. See [8] for more constraints and explanations.

Formally, the schema of a cube *schema*(C), is a tuple, say $[D_1.L_1, ..., D_n.L_n, M_1, ..., M_m]$, or simply $[L_1, ..., L_n, M_1, ..., M_m]$, with the combination of the dimension levels (each coming from a different dimension) acting as primary key and context for the measurements and a set of measures as placeholders for the (aggregate) measurements.

3.2. Cube Queries

Queries. A cube query is a cube too, specified by: (a) the detailed cube over which it is imposed, (b) a selection condition that isolates the facts that qualify for further processing, (c) the grouping levels, which determine the coarseness of the result, and (d) an aggregation over some or all measures of the cube that accompanies the grouping levels in the final result.

$$q = < C^0, \phi, [L_1, ..., L_n, M_1, ..., M_m], [agg_1(M_1^0), ...,agg_m(M_m^0)] >$$

We assume selection conditions, which are conjunctions of atomic filters of the form $L = value$ or, in general $L \in \{v_1, \dots, v_k\}$.

Constraints. In the context of this work, we imply several assumptions that mainly serve the purpose of clarity. First, we work with cube queries that involve a single measure. Second, we assume strictly two aggregator levels for the result, and third, for all atomic filters, we assume that if the dimension with a filter is also a grouper, the atomic filter is expressed in a level greater than or equal to the grouper. Then, we will employ the following query expression:

$$q = < C^0, \phi, [L_\alpha, L_\beta, M], agg(M^0) >$$

4. The Analyze Operator

In this section, we introduce the ANALYZE operator in terms of semantics, syntax, and execution strategies.

4.1. Syntax & Semantics

Assume the setup already discussed in Section 3, with a detailed cube C^0 defined over a list of hierarchical dimensions. The syntax of the ANALYZE operator is as follows:

ANALYZE $agg(M^0)$ as M FROM C^0
FOR ϕ
GROUP BY L_α, L_β
AS *queryName*

with the variables participating in the definition having the obvious semantics already introduced in Section 3.

To complement this SQL-like definition, we will also employ an algebraic representation of the operator, as $analyze(C^0, \phi, [D_\alpha.L_\alpha, D_\beta.L_\beta, M], agg(M^0))$, or equivalently as:

$$analyzeOp = \langle\ C^0, \phi, [D_\alpha.L_\alpha, D_\beta.L_\beta,M], agg(M^0)\ \rangle$$

Semantics. The semantics of the operator involve the introduction of three separate groups of intermediate cube queries, which we call facilitator queries, with each group producing a set of distinct cube query results. Assuming a given ANALYZE query AQ, the query set of $AQ.queries$ is a triplet of query sets $< Q^{org}, Q^{sib}, Q^{dd} >$ and the result of AQ, $AQ.result$ is a triplet with the results of the respective queries in the three query sets.

The 3 query sets that the operator produces are as follows: (a) Q^{org} is a singleton set, comprising exactly one query q^{org}, to which we will refer as *the original query*, that computes the aggregate value for M^0 as specified by the variables of the operator. (b) Q^{sib} is a set of sibling queries, whose goal is to compare the behavior of key values in the operator definition against the behavior of their peer values. (c) Q^{dd} is a set of drill-down queries, whose goal is to provide more detailed data to the analyst for the values produced by the original query. Figure 1 demonstrates the queries of the reference example, along with the auxiliary queries that the subsequent MQO algorithms will produce. In the sequel, we present the semantics and rationale of each of these auxiliary, facilitator queries.

4.1.1. Sibling queries

Sibling queries aim to compare (a) the different slices of the data space that pertain to the original query against (b) their peers, in order to put them in context. By comparing data slices to their peers, the data slices are effectively assessed by the analyst and, thus, contextualized.

Definition of a convenient variant for the sibling computation. A clear problem here is that if a value used in a selection atom has too many sibling values, this produces a large number of queries, even if we create siblings only for just this dimension. The problem generalizes if we take all selection atoms into consideration. To avoid this complexity, we simplify sibling generation by (i) considering siblings only for the atoms ϕ_α and ϕ_β, and (ii) by merging all sibling slices into one, by slightly adapting the aggregation level.

Given the original query q^{org}

$$q^{org} = \langle\, C^0, \phi_\alpha \wedge \phi_\beta \wedge \phi_\square, [D_\alpha.L^\gamma_\alpha, D_\beta.L^\gamma_\beta, M], agg(M^0) \,\rangle$$

we generate two sibling queries, each for one of the groupers as:

$$q^{s^\alpha} = \langle\, C^0, \phi^\star_\alpha \wedge \phi_b \wedge \phi_\square, [D_\alpha.L^\sigma_\alpha, D_\beta.L^\gamma_\beta, M], agg(M^0) \,\rangle, \phi^\star_\alpha\colon\ L^\sigma_{\alpha+1} = anc^{L^\sigma_{\alpha+1}}_{L^\sigma_\alpha}(v_\alpha)$$

$$q^{s^\beta} = \langle\, C^0, \phi_\alpha \wedge \phi^\star_\beta \wedge \phi_\square, [D_\alpha.L^\gamma_\alpha, D_\beta.L^\sigma_\beta, M], agg(M^0) \,\rangle, \phi^\star_\beta\colon\ L^\sigma_{\beta+1} = anc^{L^\sigma_{\beta+1}}_{L^\sigma_\beta}(v_\beta)$$

4.1.2. Drill-Down queries

Apart from contextualizing the result of the original query against its peers, we can also provide further details for the displayed data by drilling into the dimension hierarchies of the produced results. Ideally, this requires drilling into each of the cells of the results of the original query – see [32] on how this was traditionally done. However, this produces a significantly large number of queries to execute. We adopt a simple solution to this problem by drilling into each of the aggregator levels, by going down one level in their hierarchy. Thus, we produce two drill down queries, each drilling a level down into the detail of one of the grouper dimensions.

Given the original query q^{org}

$$q^{org} = \langle\, C^0, \phi, [D_\alpha.L^\gamma_\alpha, D_\beta.L^\gamma_\beta, M], agg(M^0) \,\rangle$$

we generate two drill-down queries, each for one of the groupers as:

$$q^{dd^{\alpha}} = \langle\, C^0,\, \phi,\, [D_{\alpha}.L^{Y}_{\alpha-1},\, D_{\beta}.L^{Y}_{\beta}, M^{dd^{\alpha}}],\, agg(M^0)\,\rangle$$

$$q^{dd^{\beta}} = \langle\, C^0,\, \phi,\, [D_{\alpha}.L^{Y}_{\alpha},\, D_{\beta}.L^{Y}_{\beta-1}, M^{dd^{\beta}}],\, agg(M^0)\,\rangle$$

---

5. Multiple Query Optimization for Analyze Queries

So far, we have seen that the ANALYZE operator is actually a composition of 5 facilitator queries to the underlying database. In this section, we introduce a method that replaces the need of issuing five queries to the database with a merging of (a subset of) them, along with algorithms that exploit this merging.

5.1. Theoretical Background

We base our theoretical construction on the Cube Usability Theorem [8, 39], which states that it is possible to derive the result of a cube query by filtering already accessed data by another cube query and re-aggregate them, if certain conditions are held.

In our case, we introduce the auxiliary, *all-encompassing query* q^A for the ANALYZE operator. The all-encompassing query q^A is characterized by: (0) the same basic cube and measure aggregation, (1) groupers involving the highest levels of aggregation that are low enough to answer any of the internal queries of the ANALYZE operator, and (2) the broadest possible selection condition to encompass all the detailed tuples needed to answer any internal query. In the rest of this section we will first show that the all-encompassing query q^A can answer all the internal queries, and, second, we will accompany the feasibility result with two algorithms that actually compute the answer.

Theorem 5.1 (Multi-Query Usability)**.** *Assume the query:*
analyze $= \langle\, C^0,\, \phi_{\alpha} \wedge \phi_{\beta} \wedge \phi_{\square},\, [D_{\alpha}.L^{Y}_{\alpha},\, D_{\beta}.L^{Y}_{\beta}, M],\, agg(M^0)\,\rangle$
producing five internal queries as prescribed in the definition of the ANALYZE operator. The following all-encompassing query q^A *can answer all the internal queries of the ANALYZE operator.*

$$q^A = \langle C^0,\, \phi^{\star}_{\alpha} \wedge \phi^{\star}_{\beta} \wedge \phi_{\square},\, [L^{Y}_{\alpha-1}, L^{Y}_{\beta-1}, L^{Y}_{\alpha}, L^{Y}_{\beta}, L^{\sigma}_{\alpha}, L^{\sigma}_{\beta}, M^A],\, agg(M^0)\rangle,$$
$$\text{with } \phi^{\star}_{\alpha}\colon L^{\sigma}_{\alpha+1} = anc^{L^{\sigma}_{\alpha+1}}_{L^{\sigma}_{\alpha}}(v_{\alpha}) \text{ and } \phi^{\star}_{\beta}\colon L^{\sigma}_{\beta+1} = anc^{L^{\sigma}_{\beta+1}}_{L^{\sigma}_{\beta}}(v_{\beta})$$

5.2. The Max Multi-Query Optimization Algorithm

The **MaxMQO Algorithm** describes how to compute all facilitator queries of the ANALYZE operator with a single access to the underlying database. The algorithm takes as input the definition of the original query, which is sufficient to determine the entire set of facilitator queries and to produce their results as output.

Step 1. First, the algorithm constructs five maps, one for each of the queries that determine the ANALYZE operator. Each of these maps will store the result of the respective query. The key of the map is the combination of grouper values (which are pretty much the coordinates of each result cell), and the value is the aggregate measure that pertains to these coordinates.

Step 2. Second, the algorithm constructs and executes the all-encompassing auxiliary query q^A that will serve as the basis to compute all the other query results.

Algorithm 1: Algorithm **Max** MQO

Input: Original query $q^{org} = \langle\, C^0, \phi, [D_\alpha.L^\gamma_\alpha, D_\beta.L^\gamma_\beta, M], agg(M^0)\,\rangle$, *phi*: $\phi_\alpha \wedge \phi_\beta \wedge \phi_\square$, phi_i : $D_i.L^\sigma_i = v_i, i : \alpha \text{ or } \beta$

Output: Updated results for q^{org}, q^{s^A}, q^{s^B}, q^{dd^A}, q^{dd^B}

begin

  Let $H^{org}, H^{s^A}, H^{s^B}, H^{dd^A}, H^{dd^B}$ be maps of the form $H : \langle\, dom(L^\gamma_1), dom(L^\gamma_2)\,\rangle \rightarrow dom(M^0)$;

  Let $aggF^\star = adapter(aggF)$;

  Let $\phi^\star_\alpha$: $L^\sigma_{\alpha+1} = anc^{L^\sigma_{\alpha+1}}_{L^\sigma_\alpha}(v_\alpha)$ and $\phi^\star_\beta$: $L^\sigma_{\beta+1} = anc^{L^\sigma_{\beta+1}}_{L^\sigma_\beta}(v_\beta)$;

  Let $q^A = \langle\, C^0, \phi^\star_\alpha \wedge \phi^\star_\beta \wedge \phi_\square, [L^\gamma_{\alpha-1}, L^\gamma_{\beta-1}, L^\gamma_\alpha, L^\gamma_\beta, L^\sigma_\alpha, L^\sigma_\beta, M^A], agg(M^0)\,\rangle$;

  $q^A.cells = q^A.execute()$;

  foreach *tuple t in the result set* $q^A.cells$ **do**

    Assume $t = \langle v^\gamma_{\alpha-1}, v^\gamma_{\beta-1}, v^\gamma_\alpha, v^\gamma_\beta, v^\sigma_\alpha, v^\sigma_\beta, m\rangle$;

    if $\sigma_{\phi_\alpha \wedge \phi_\beta}(t)$ **then**

      $updateMap(H^{org}, \langle\langle v^\gamma_\alpha, v^\gamma_\beta\rangle, m\rangle, agg)$;

      $updateMap(H^{dd^A}, \langle\langle v^\gamma_{\alpha-1}, v^\gamma_\beta\rangle, m\rangle, agg)$;

      $updateMap(H^{dd^B}, \langle\langle v^\gamma_\alpha, v^\gamma_{\beta-1}\rangle, m\rangle, agg)$;

    end

    if $\sigma_{\phi_\beta}(t)$ **then**

      $updateMap(H^{s^A}, \langle\langle v^\sigma_\alpha, v^\gamma_\beta\rangle, m\rangle, agg)$;

    end

    if $\sigma_{\phi_\alpha}(t)$ **then**

      $updateMap(H^{s^B}, \langle\langle v^\gamma_\alpha, v^\sigma_\beta\rangle, m\rangle, agg)$;

    end

  end

  $q^{org}.cells = H^{org}$; $q^{dd^A}.cells = H^{dd^A}$, $q^{dd^B}.cells = H^{dd^B}$; $q^{s^A}.cells = H^{s^A}$; $q^{s^B}.cells = H^{s^B}$;

  return $\langle q^{org}.cells, q^{s^A}.cells, q^{s^B}.cells, q^{dd^A}.cells, q^{dd^B}.cells\rangle$;

end

Function `updateMap`(H : *map*, $\langle\langle g_1, g_2\rangle, m\rangle$: $Tuple\langle Pair\langle grouper, grouper\rangle, measure\rangle$, $aggF^\star$: *agg. Func.*)**:**

  if $H.containsKey(\langle g_1, g_2\rangle)$ **then**

    $curValue = H.get(\langle g_1, g_2\rangle)$; $H.put(\langle g_1, g_2\rangle, aggF^\star(currValue, m))$;

  end

  else

    $H.put(\langle g_1, g_2\rangle, m)$;

  end

Step 3. Once the tuples of q^A have been computed, we need to distribute them to the facilitator queries of the operator. We visit each tuple at a time. All tuples will end up in the siblings, as the selection condition of q^A is exactly the one of the siblings. Remember that the selection condition of the siblings is broader than the one of the original query and the drill-downs (which is identical to the original one). However, some of the resulting tuples also pertain to the original and the drill-down queries, and so, they have to be pushed towards the respective maps. In the end, the hash-maps contain the results of the auxiliary queries.

Algorithm 2: Algorithm **Mid** MQO

Input: Original query $q^{org} = \langle C^0, \phi, [D_\alpha.L^Y_\alpha, D_\beta.L^Y_\beta, M], agg(M^0)\rangle$,
$\phi = \phi_\alpha \wedge \phi_\beta \wedge \phi_\square, \quad \phi_i : D_i.L^\sigma_i = v_i,\ i \in \{\alpha, \beta\}$

Output: Updated results for $q^{org}, q^{s^A}, q^{s^B}, q^{dd^A}, q^{dd^B}$

begin
  Let $H^{org}, H^{s^A}, H^{s^B}, H^{dd^A}, H^{dd^B}$ be maps of the form
    $H : \langle dom(L^Y_1), dom(L^Y_2)\rangle \rightarrow dom(M^0)$;
  Let $aggF^\star = adapter(aggF)$;
  `/* Construct and execute Mid MQO queries */`
  Let $\phi^*_\alpha : L^\sigma_{\alpha+1} = anc_{L^\sigma_{\alpha+1}L^\sigma_\alpha}(v_\alpha)$, Let $\phi^*_\beta : L^\sigma_{\beta+1} = anc_{L^\sigma_{\beta+1}L^\sigma_\beta}(v_\beta)$;
  Let $q^{org\&dd} = \langle C^0, \phi_\alpha \wedge \phi_\beta, [L^Y_{\alpha-1}, L^Y_{\beta-1}, L^Y_\alpha, L^Y_\beta, M^{org\&dd}], agg(M^0)\rangle$;
  Let $q^{s^A} = \langle C^0, \phi^*_\alpha \wedge \phi_\beta, [L^Y_{\alpha+1}, L^Y_\beta, M^{s^A}], agg(M^0)\rangle$;
  Let $q^{s^B} = \langle C^0, \phi_\alpha \wedge \phi^*_\beta, [L^Y_\alpha, L^Y_{\beta+1}, M^{s^B}], agg(M^0)\rangle$;
  $q^{org\&dd}.cells \leftarrow q^{org\&dd}.execute()$;
  $q^{s^A}.cells \leftarrow q^{s^A}.execute()$, $q^{s^B}.cells \leftarrow q^{s^B}.execute()$;
  foreach *tuple* $t \in q^{org\&dd}.cells$ **do**
    Assume $t = \langle v^Y_{\alpha-1}, v^Y_{\beta-1}, v^Y_\alpha, v^Y_\beta, m\rangle$;
    if $\sigma_{\phi_\alpha \wedge \phi_\beta}(t)$ **then**
      $updateMap(H^{org}, \langle\langle v^Y_\alpha, v^Y_\beta\rangle, m\rangle, agg)$;
      $updateMap(H^{dd^A}, \langle\langle v^Y_{\alpha-1}, v^Y_\beta\rangle, m\rangle, agg)$;
      $updateMap(H^{dd^B}, \langle\langle v^Y_\alpha, v^Y_{\beta-1}\rangle, m\rangle, agg)$;
    end
  end
  $q^{org}.cells \leftarrow H^{org}$, $q^{dd^A}.cells \leftarrow H^{dd^A}$, $q^{dd^B}.cells \leftarrow H^{dd^B}$;
  return $\langle q^{org}.cells, q^{s^A}.cells, q^{s^B}.cells, q^{dd^A}.cells, q^{dd^B}.cells\rangle$;
end
Function `updateMap`$(H : map, \langle\langle g_1, g_2\rangle, m\rangle :$
$Tuple\langle Pair\langle grouper, grouper\rangle, measure\rangle, aggF^\star)$**:**
  if $H.containsKey(\langle g_1, g_2\rangle)$ **then**
    $curValue \leftarrow H.get(\langle g_1, g_2\rangle)$;
    $H.put(\langle g_1, g_2\rangle, aggF^\star(curValue, m))$;
  end
  else
    $H.put(\langle g_1, g_2\rangle, m)$;
  end

5.3. Mid - Multi Query Optimization Algorithm for the Analyze Operator

The Max Multi-Query Optimization Algorithm utilizes a single query on the underlying database to retrieve the result of an *ANALYZE* query. However, the selection conditions of the all-encompassing auxiliary query q^A explore a vast super-set of tuples that contain the query result for each of the five queries, plus tuples that are not part of any query result, thus they are not aggregated. The spare tuples add performance latency, since they have to be processed by Max MQO, in order to determine whether a tuple belongs to one of the five query result maps.

To reduce the vastness of the explored data space, we can reduce the number of spare tuples returned by q^A. Observe that the selection conditions of the original query and the drill-down queries are the

same. We take advantage of that property to construct a single query q^{MID} that combines the original and drill-down facilitator queries, without touching the siblings.

The **Mid MQO Algorithm** launches the two sibling queries along with the merged q^{MID} query. Since the latter pertains to three facilitator queries, we group the tuples on their original and ancestor levels to distribute the result tuples. In that way, we get the tuples that contribute to the original and drill-down queries result in a single access. The rest of the processing is the same as with Max MQO. Compared to q^A, q^{MID} returns less detailed tuples and applies less groupers to them, thus reducing the result size.

6. Experiments

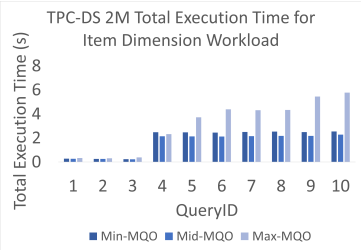
In this section, we experimentally evaluate the execution time of the proposed algorithms under varying configurations of data and query workloads. In all of our experiments, we use the Delian Cubes system [40], which is a cube query answering engine, within which we have incorporated our algorithms. We have employed MySQL 8.0.34 as the underlying database. For our evaluation, the methods run on a Windows 11 2.50 GHz 14-core processor system with 32GB main memory and 1TB SSD. All the material is found at: https://github.com/DAINTINESS-Group/DelianCubeEngine.

6.1. Experimental Setup

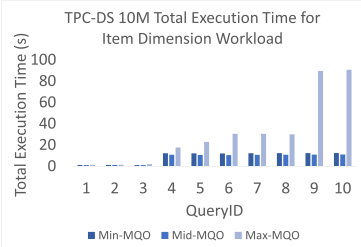
Experimental Goal and Evaluation Metrics. The main goal of our evaluation is to evaluate the efficiency of the alternative algorithms for executing the ANALYZE operator. Therefore, for all antagonists, we measure the *Total Execution Time* needed to fully compute the answer to an ANALYZE query, as well as its breakdown to different facilitators within each algorithm.

Competitor Algorithms. In our experiments, we compare the three different algorithms to implement the *ANALYZE* operator over *Total Execution Time* and its breakdown. All algorithms receive as input an ANALYZE query with two groupers, a distributive aggregate function, and at least two selection conditions. *Min-MQO* is the algorithm that performs the least merging of the operator's underlying cube queries. Practically, this is the plain simple execution of the operator without any optimization. *Max-MQO* is the algorithm that performs the maximum merging of the facilitator queries into a single facilitator query with four extra groupers to cover the *siblings* and *drill-down* results, substituted selection conditions. The results are post-processed and the tuples are distributed to the correct placeholder with respect to the operator semantics. *Mid-MQO* is the algorithm that strikes a balance in the middle of the two extremes of full- and no- merging of the facilitator queries into one. Specifically, the algorithm constructs three facilitator queries: (i) two *sibling* queries, one for each grouping dimension, and, (ii) a merged *original-n-drill-down* multi-query which is a combination of the *original* and *drill-down* queries.

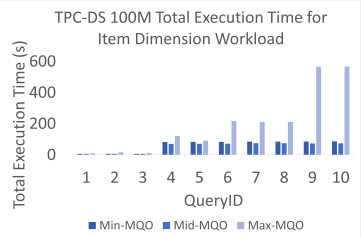
Datasets We have employed the datasets listed below to perform our evaluation. Specifically, these datasets are: (i) *Northwind* [41], which is a synthetic dataset that represents the sales and the operations of a food import/export company, (ii) *Foodmart*, which is a cubefied version of *Foodmart* [42], which contains synthetic data regarding the sales activity of a retail supermarket, (iii) *pkdd99+*, which is an upscaled version of *pkdd99* [43], a financial dataset that contains data about loans and transactions, for which we have created a fact table that contains 100 million entries, and, (iv) *TPC-DS* [44], which is an industry standard benchmark designed to evaluate the performance of analytical platforms. We have tested the scalability of our algorithms over three versions of *TPC-DS* with scale factor 1, 3.5, and, 35. As Delian Cubes operates over data cubes, we do not directly utilize the raw schema of the datasets, but rather adapt it to a cubefied, clean, star-schema version, in order to support hierarchies and cube queries.



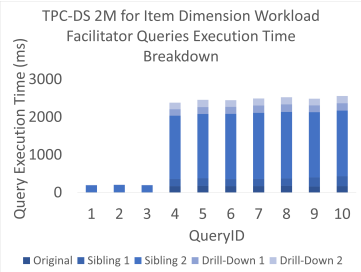
(a) TPC-DS 2M *Item* Workload



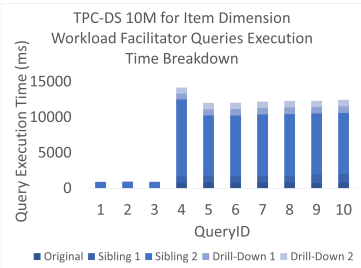
(b) TPC-DS 10M *Item* Workload



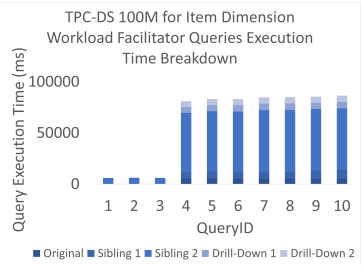
(c) TPC-DS 100M *Item* Workload



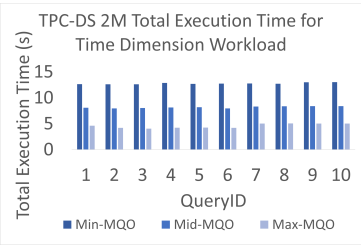
(d) TPC-DS 2M *Item* Workload Facilitator Queries Breakdown



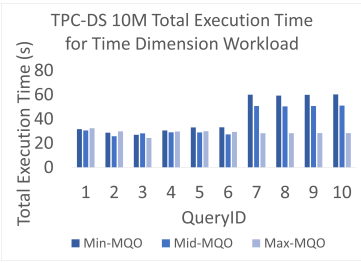
(e) TPC-DS 10M *Item* Workload Facilitator Queries Breakdown



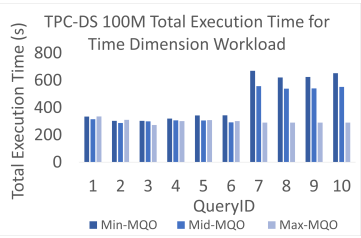
(f) TPC-DS 100M *Item* Workload Facilitator Queries Breakdown



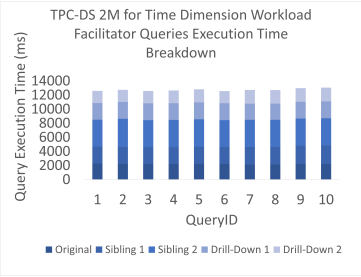
(g) TPC-DS 2M *Time* Workload



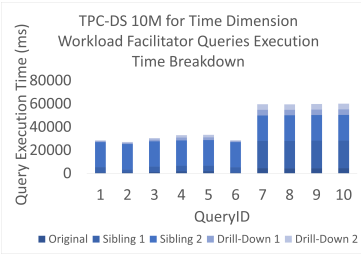
(h) TPC-DS 10M *Time* Workload



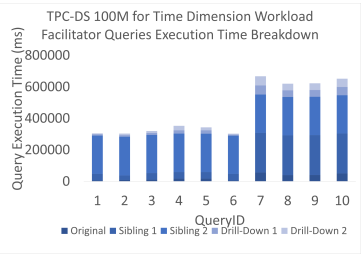
(i) TPC-DS 100M *Time* Workload



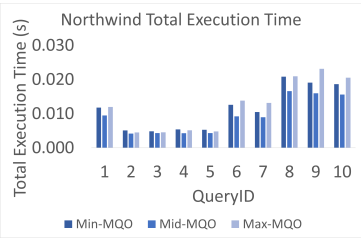
(j) TPC-DS 2M *Time* Workload Facilitator Queries Breakdown



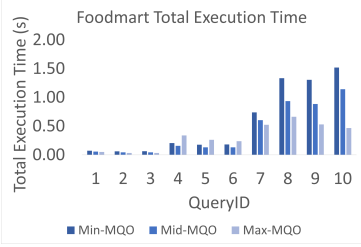
(k) TPC-DS 10M *Time* Workload Facilitator Queries Breakdown



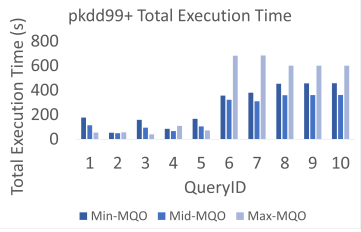
(l) TPC-DS 100M *Time* Workload Facilitator Queries Breakdown



(m) Northwind Workload



(n) Foodmart Time Workload



(o) pkdd99+ Time Workload

Figure 2: Performance Evaluation Figures

Query Workloads. To assess the operator with different selectivities over the fact tables, for each data set, we use a workload of 10 queries where we (a) vary with the various levels of filters/groupers, and (b) employ one large and one small dimension table. For the TPC-DS dataset, and in order to also evaluate the effect of dimension table size, we use two query workloads, both involving the large *Date* dimension: (i) a workload using the large *Time* dimension table, and (b) a workload using the small *Item* dimension table. In all workloads, as the QueryID increases, the selectivity of q^{org} also increases in all data sets.

6.2. Algorithm Evaluation

Performance Evaluation. Figures 2a, 2b, and 2c illustrate the *Item* query workload execution time across all methods for each TPC-DS dataset size variation. Min-MQO shown as a dark blue line, Mid-MQO as a blue line, Max-MQO as a light blue line. The dimension tables involved in the workload are the *Date* dimension (≈80000 tuples) and the *Item* dimension (≈ 20000 tuples). We observe the following: (i) Regardless of the dataset size, Mid-MQO performs the best, (ii) Max-MQO constantly performs the worst, especially when the selectivity ratio is increasing, and (iii) Min-MQO and Mid-MQO do not differ significantly in their execution time. Figures 2d,2e,2f provide the detailed execution time for each facilitator query of the workload's ANALYZE queries. We observe that the drill-down queries that Mid-MQO does not execute in comparison to Min-MQO, are not as significant for the total execution time. Thus, the difference in performance in Min-MQO and Mid-MQO is relatively small. Max-MQO processes a large number of tuples compared to the other antagonists and, in that context, underperforms constantly.

Figures 2g, 2h, 2i illustrate the *Time* query workload execution time across all methods for each TPC-DS dataset size variation. The dimension tables involved in the workload are the *Date* dimension (≈80000 tuples) and the *Time* dimension (≈ 70000 tuples). We observe the following: (i) In the TPC-DS 2M, Max-MQO outperforms the other algorithms and Min-MQO is the worst performing algorithm in every case, (ii) in the TPC-DS 10M and 100M, in small selectivity ratios, Max-MQO performance is almost stable regarding the selectivity ratio increase, while Min-MQO's and Mid-MQO's performance deteriorates as the selectivity ratio increases. Here, due to the large size of both dimension tables, the performance of Max-MQO is stable, as it consistently explores a large subset of the fact table once, and avoids the heavy extra cost of siblings of the other algorithms when the selectivity increases (see Figures 2j, 2k, 2l for the breakdown, especially for high-selectivity queries with large QueryID's).

Finally, when comparing the algorithms over the 3 other datasets (Figure 2m, 2n, 2o) we observe: (i) in the Northwind workload that involves two small dimension tables (≈90 tuples and ≈9 tuples), Mid-MQO constantly scores wins against the other antagonists, while Max-MQO is underperforming, (ii) in the Foodmart workload that involves unbalanced dimension tables (≈10000 tuples and ≈1000 tuples), Max-MQO is the fastest algorithm, while Min-MQO is the slowest, (iii) in the pkdd99+ workload that involves one fairly large and one small dimension (≈30000 tuples and ≈5000), Mid-MQO outperforms its antagonists in the high selectivity ratios, while Max-MQO performs better in low selectivity ratios.

We have also conducted extensive experiments over the number of atomic filters, the level of groupers etc, not reported here for lack of space, but not significantly affecting performance in the same extent as the selectivity and the data size [9].

Choosing algorithms. *Mid-MQO is a safe choice as a query processing strategy, as it is both stable and wins most of the time*. Can we do better than this first result, though? We have analyzed the entire corpus of 70 workloads of the large TPC-DS and pkdd99+ datasets (7 workloads of 10 queries each) to *precisely detect* when each query processing strategy wins, via easily computable cost measures. Ultimately (Figure 3), *Max-MQO wins when (a) the siblings touch a fairly large number of fact tuples compared to the all-encompassing query (more than 40%), and, (b) there is a fairly small imbalance between the sibling queries (less than 45%). In all other cases, Mid-MQO wins.* The rationale is simple: (1) when q^A is too large compared to the facilitator queries, then, it explores too much of the data space, hence Max-MQO loses; (2) when the siblings are both sizable and imbalanced, they do not significantly overlap,

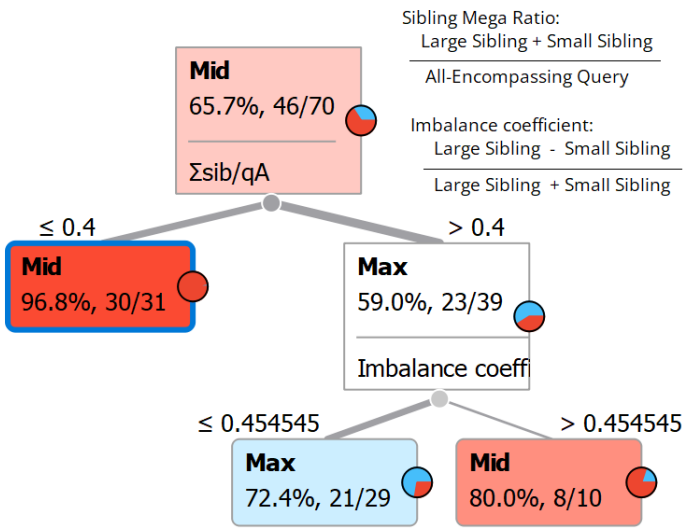


Figure 3: Criteria for selecting the best algorithm

thus Max-MQO, visiting them once, does not save time. The only case where Max-MQO saves time is if sizable siblings overlap (estimated via the imbalance coefficient). Naturally, the exact thresholds are an engineering default, subject to wider evaluation over time, however, the rationale is consistent with the theoretical setup of q^A.

7. Conclusions

The paper introduces ANALYZE, a novel intentional cube query operator that provides a 360° view of the data by combining original, sibling, and drill-down queries in a single invocation with formal semantics. We have shown that the operator's internal queries can be safely merged with theoretical correctness guarantees, enabling principled multi-query optimization at the operator level without modifying the underlying query optimizer. We have introduced, implemented in a data analytics system, and extensively evaluated three execution strategies, demonstrating that partial merging (Mid-MQO) delivers robust and scalable performance across workloads, full merging (Max-MQO) is beneficial only when the siblings are sizable and significantly overlap, while the not-optimized strategy (Min-MQO) is consistently suboptimal.

Right now, a plain DBMS would simply execute Min-MQO to process the set of facilitator queries of the operator. A lesson learned here is that the external data analytics engine *can* optimize query execution *outside the DBMS*. This is a lesson to be applied to future intentional operators as well (PREDICT, EXPLAIN, etc). Exploiting the formal, algebraic nature of the operator also raises the question of integrating it with cost-based DBMS optimizers or platforms like Apache Spark. Relaxing some of the constraints (e.g., via more groupers, siblings based on user-defined or context-dependent benchmarks [45, 46]) is also open to research.

Acknowledgments

The research project is implemented in the framework of H.F.R.I call "3rd Call for H.F.R.I.'s Research Projects to Support Faculty Members & Researchers" (H.F.R.I. Project Number: 23640).

D. Gkitsakis helped with early, trial versions of the code.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] S. Sarawagi, Explaining differences in multidimensional aggregates, in: VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK, 1999, pp. 42–53.

[2] S. Sarawagi, R. Agrawal, N. Megiddo, Discovery-driven exploration of OLAP data cubes, in: Advances in Database Technology - EDBT'98, 6th International Conference on Extending Database Technology, Valencia, Spain, March 23-27, 1998, Proceedings, 1998, pp. 168–182.

[3] Y. Wang, Z. Sun, H. Zhang, W. Cui, K. Xu, X. Ma, D. Zhang, Datashot: Automatic generation of fact sheets from tabular data, IEEE Trans. Vis. Comput. Graph. 26 (2020) 895–905.

[4] F. Abuzaid, P. Kraft, S. Suri, E. Gan, E. Xu, A. Shenoy, A. Ananthanarayan, J. Sheu, E. Meijer, X. Wu, J. F. Naughton, P. Bailis, M. Zaharia, DIFF: a relational interface for large-scale data explanation, VLDB J. 30 (2021) 45–70.

[5] P. Ma, R. Ding, S. Han, D. Zhang, Metainsight: Automatic discovery of structured knowledge for exploratory data analysis, in: SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021, ACM, 2021, pp. 1262–1274.

[6] P. Vassiliadis, P. Marcel, The road to highlights is paved with good intentions: Envisioning a paradigm shift in OLAP modeling, in: Proceedings of the 20th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data co-located with 10th EDBT/ICDT Joint Conference (EDBT/ICDT 2018), Vienna, Austria, March 26-29, 2018, 2018.

[7] P. Vassiliadis, P. Marcel, S. Rizzi, Beyond roll-up's and drill-down's: An intentional analytics model to reinvent OLAP, Information Systems 85 (2019) 68–91.

[8] P. Vassiliadis, A Cube Algebra with Comparative Operations: Containment, Overlap, Distance and Usability, CoRR abs/2203.09390 (2022).

[9] M. Iakovidis, P. Vassiliadis, Semantics and multi-query optimization algorithms for the analyze operator, CoRR abs/2602.08546 (2026).

[10] T. Milo, A. Somech, Automating exploratory data analysis via machine learning: An overview, in: Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020, 2020, pp. 2617–2622.

[11] S. Sarawagi, User-adaptive exploration of multidimensional data, in: VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt, 2000, pp. 307–316.

[12] G. Sathe, S. Sarawagi, Intelligent rollups in multidimensional OLAP data, in: VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy, 2001, pp. 531–540.

[13] S. Idreos, O. Papaemmanouil, S. Chaudhuri, Overview of data exploration techniques, in: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, 2015, pp. 277–281.

[14] B. Tang, S. Han, M. L. Yiu, R. Ding, D. Zhang, Extracting top-k insights from multi-dimensional data, in: Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017, 2017, pp. 1509–1524.

[15] O. B. El, T. Milo, A. Somech, ATENA: an autonomous system for data exploration based on deep reinforcement learning, in: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019, 2019, pp. 2873–2876.

[16] R. Ding, S. Han, Y. Xu, H. Zhang, D. Zhang, QuickInsights: Quick and automatic discovery of insights from multi-dimensional data, in: Proceedings of SIGMOD, 2019, pp. 317–332.

[17] A. Personnaz, S. Amer-Yahia, L. Berti-Équille, M. Fabricius, S. Subramanian, DORA THE EX-

PLORER: exploring very large data with interactive deep reinforcement learning, in: CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021, 2021, pp. 4769–4773.
[18] D. Shi, X. Xu, F. Sun, Y. Shi, N. Cao, Calliope: Automatic visual data story generation from a spreadsheet, IEEE Trans. Vis. Comput. Graph. 27 (2021) 453–463.
[19] T. D. Bie, L. D. Raedt, J. Hernández-Orallo, H. H. Hoos, P. Smyth, C. K. I. Williams, Automating data science, Commun. ACM 65 (2022) 76–87.
[20] S. Amer-Yahia, P. Marcel, V. Peralta, Data narration for the people: Challenges and opportunities, in: Proceedings 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023, OpenProceedings.org, 2023, pp. 855–858.
[21] M. Sun, L. Cai, W. Cui, Y. Wu, Y. Shi, N. Cao, Erato: Cooperative data story editing via fact interpolation, IEEE Trans. Vis. Comput. Graph. 29 (2023) 983–993.
[22] H. Li, L. Ying, H. Zhang, Y. Wu, H. Qu, Y. Wang, Notable: On-the-fly assistant for data storytelling in computational notebooks, in: CHI, 2023.
[23] P. Ma, R. Ding, S. Wang, S. Han, D. Zhang, Insightpilot: An llm-empowered automated data exploration system, in: EMNLP'2023, 2023.
[24] S. Amer-Yahia, Intelligent agents for data exploration, Proc. VLDB Endow. 17 (2024) 4521–4530.
[25] J. Xing, X. Wang, H. V. Jagadish, Data-driven insight synthesis for multi-dimensional data, VLDB Endow. 17 (2024) 1007–1019.
[26] T. Lipman, T. Milo, A. Somech, T. Wolfson, O. Zafar, LINX: A language driven generative system for goal-oriented automated data exploration, in: Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025, OpenProceedings.org, 2025, pp. 270–283.
[27] M. Francia, M. Golfarelli, P. Marcel, S. Rizzi, P. Vassiliadis, Suggesting assess queries for interactive analysis of multidimensional data, IEEE Trans. Knowl. Data Eng. 35 (2023) 6421–6434.
[28] M. Francia, M. Golfarelli, P. Marcel, S. Rizzi, P. Vassiliadis, Assess queries for interactive analysis of data cubes, in: Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021, 2021, pp. 121–132.
[29] M. Francia, P. Marcel, V. Peralta, S. Rizzi, Enhancing cubes with models to describe multidimensional data, Inf. Syst. Frontiers 24 (2022) 31–48.
[30] M. Francia, S. Rizzi, P. Marcel, Explaining cube measures through intentional analytics, Inf. Syst. 121 (2024) 102338.
[31] D. Gkesoulis, P. Vassiliadis, Cinecubes: cubes as movie stars with little effort, in: Proceedings of the sixteenth international workshop on Data warehousing and OLAP, DOLAP 2013, San Francisco, CA, USA, October 28, 2013, 2013, pp. 3–10.
[32] D. Gkesoulis, P. Vassiliadis, P. Manousis, Cinecubes: Aiding data workers gain insights from OLAP queries, Inf. Syst. 53 (2015) 60–86.
[33] T. K. Sellis, Multiple-query optimization, ACM Trans. Database Syst. 13 (1988) 23–52.
[34] T. K. Sellis, S. Ghosh, On the multiple-query optimization problem, IEEE Trans. Knowl. Data Eng. 2 (1990) 262–266.
[35] P. Roy, S. Sudarshan, Multi-query optimization, in: Encyclopedia of Database Systems, Springer US, 2009, pp. 1849–1852.
[36] M. Hong, M. Riedewald, C. Koch, J. Gehrke, A. J. Demers, Rule-based multi-query optimization, in: EDBT 2009, 12th International Conference on Extending Database Technology, Saint Petersburg, Russia, March 24-26, 2009, Proceedings, 2009, pp. 120–131.
[37] W. Le, A. Kementsietsidis, S. Duan, F. Li, Scalable multi-query optimization for SPARQL, in: IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012, 2012, pp. 666–677.
[38] T. Kathuria, S. Sudarshan, Greedy awakens : Efficient and provable multi-query optimization, CoRR abs/1512.02568 (2015).
[39] P. Vassiliadis, Cube query answering via the results of previous cube queries, in: Proceedings of the 25th International Workshop on Design, Optimization, Languages and Analytical Processing of

Big Data (DOLAP) co-located with the 26th International Conference on Extending Database Technology and the 26th International Conference on Database Theory (EDBT/ICDT 2023), Ioannina, Greece, March 28, 2023, CEUR-WS.org, 2023, pp. 71–75.

[40] Delian Cubes, https://github.com/DAINTINESS-Group/DelianCubeEngine, 2025. DAINTINESS GROUP University of Ioannina.

[41] Microsoft, Northwind, https://github.com/microsoft/sql-server-samples/tree/master/samples/databases/northwind-pubs, 2023. Accessed: 2025-09-14.

[42] J. Hyde, Foodmart, https://github.com/julianhyde/foodmart-data-hsqldb, 2025. Accessed: 2025-07-24.

[43] N. Zhong, Y. Yao, S. Ohsuga, Peculiarity oriented multi-database mining, in: Principles of Data Mining and Knowledge Discovery, Third European Conference, PKDD '99, Prague, Czech Republic, September 15-18, 1999, Proceedings, volume 1704 of *Lecture Notes in Computer Science*, Springer, 1999, pp. 136–146.

[44] TPC, Tpc-ds, https://www.tpc.org/tpcds/, 2024. Accessed: 2025-06-05.

[45] K. Stefanidis, E. Pitoura, P. Vassiliadis, A context-aware preference database system, Int. J. Pervasive Comput. Commun. 3 (2007) 439–460.

[46] P. Mateos, A. Bellogín, A systematic literature review of recent advances on context-aware recommender systems, Artif. Intell. Rev. 58 (2025) 20.