Data Storytelling via Sibling Queries and Highlight Extraction

A Thesis

submitted to the designated by the Assembly of the Department of Computer Science and Engineering
Examination Committee

by

Aggeliki Dougia

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN DATA AND COMPUTER SYSTEMS ENGINEERING

WITH SPECIALIZATION
IN ADVANCED COMPUTER SYSTEMS

University of Ioannina
School of Engineering
Ioannina 2024

DEDICATION

To my family & my friends

Acknowledegments

First, I would love to thank the supervisor of my master thesis, Panos Vassiliadis. He was always supportive, providing me with all possible assistance through his knowledge. However, what I will always hold dear to my heart are his encouraging words, his advice, and the kindness he showed, not only to me but to all the students he supervises. It will be an honor for me to emulate even a fraction of these virtues. Also, I would like to thank PhD candidate Dimos Gkitsakis, for all the help he provided me during my master thesis.
I would also like to express my heartfelt gratitude to all my friends and especially Aggeliki Kostadima and Konstantina Mylona. Their constant support, understanding, and encouragement have been invaluable throughout this journey.
Finally, I want to extend my deepest thanks to my parents, Lazaros and Christina, my sister, Anastasia, and my beloved uncle, Alexandros. Their love, patience, and sacrifices have been the foundation upon which I have built my academic studies. Without their belief in me, completing this thesis would not have been possible.

Ioannina, June 2024
Aggeliki Dougia

CONTENTS

LIST OF FIGURES

ABSTRACT

Aggeliki Dougia, M.Sc. in Data and Computer Systems Engineering, Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, June 2024
Thesis Title: Data Storytelling via Sibling Queries and Highlight Extraction
Advisor: Panos Vassiliadis, Professor

Business Intelligence relies on (a) hierarchical multidimensional data, coming in the form of data cubes and dimension hierarchies for the representation of information, and (b) advanced querying operators for the extraction of interesting facts from the available data.

In this Thesis, we propose a querying operator that takes a chart specification as its input and produces a data story as its output. The motivation for this *Query-As-A-Chart* operator lies in the observation that traditional queries, including Business Intelligence ones, are inadequate to express the higher-level knowledge goals that analysts currently have.

To the extent that query results are typically visualized, the operator takes this fact as input, and complements its specification with the type of graphical representation (*chart*) expected to be produced, in the form of a line-chart, bar-chart, or scatter plot. Apart from this first extension to traditional querying, in the form of extending the query specification, a second extension includes the extension of how query results are handled. In the context of this Thesis, two extensions are added to the original query execution. The first concerns complementing the original query with auxiliary, *sibling* queries that provide results of similar subsets of the data space to the original one. So, for example, if the original query concerns an atomic filter of the form "*city* = *Ioannina*", we want to automatically generate sibling queries for all the cities that pertain to the same country as *Ioannina*, i.e., *Greece*, and contrast the results. A second

extension has to do with the application of *models* to the resulting data. Assuming one of the grouper dimensions is time, thus producing a timeseries as a result of the query, models like trend, unimodality, bimodality, or dominance, can be applied over the different timeseries produced for the different queries and compared for commonalities and exceptions. The extracted *highlights* are appropriately scored, ranked and pruned on the basis of a simple *rank-n-prune* filter.

The result of the entire process is (a) a set of charts that visualize the results, but most importantly, (b) a combined report, or *data story*, that combines the graphical representations, along with a textual description of the detected highlights, to be returned to the analyst.

Εκτεταμενη περιληψη

Αγγελική Δούγια, Δ.Μ.Σ. στη Μηχανική Δεδομένων και Υπολογιστικών Συστημάτων, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων, Ιούνιος 2024
Τίτλος Διατριβής: Εξιστόρηση Δεδομένων με Όμορα Ερωτήματα και Εξαγωγή Διακεκριμένων Ευρημάτων
Επιβλέπων: Βασιλειάδης Παναγιώτης, Καθηγητής

Η Επιχειρησιακή νοημοσύνη (Business Intelligence) στηρίζεται σε α) σε ιεραρχικά πολυδιάστατα δεδομένα, που έρχονται με τη μορφή κύβων δεδομένων και διαστάσεων με ιεραρχίες για την αναπαράσταση πληροφοριών και β) προηγμένους τελεστές ερωτημάτων για την εξαγωγή ενδιαφερόντων φαινομένων από τα διαθέσιμα δεδομένα.
Σε αυτή την εργασία, προτείνουμε έναν τελεστή ερωτήματος που παίρνει τον προσδιορισμό ενός γραφήματος ως είσοδο και παράγει μια ιστορία δεδομένων (data story) ως έξοδο. Το κίνητρο για αυτόν τον τελεστή Query-As-A-Chart έγκειται στην παρατήρηση ότι τα παραδοσιακά ερωτήματα, συμπεριλαμβανομένων αυτών της επιχειρησιακής νοημοσύνης, είναι ανεπαρκή για να εκφράσουν τους στόχους γνώσης υψηλότερου επιπέδου που έχουν επί του παρόντος οι αναλυτές.
Στην περίπτωση που τα αποτελέσματα του ερωτήματος μπορούν να οπτικοποιηθούν, ο τελεστής λαμβάνει το ερώτημα ως είσοδο μαζί με τον προσδιορισμό της γραφικής παράστασης (γραφήματος) που επέλεξε ο αναλυτής για να δει. Οι διαθέσιμες επιλογές είναι γραμμικό γράφημα (line-chart), ραβδόγραμμα (bar-chart) και γράφημα διασποράς (scatter-plot).
Εκτός από την πρώτη επέκταση στην παραδοσιακή αναζήτηση, με τον προσδιορισμό γραφήματος μαζί με το ερώτημα, μια δεύτερη επέκταση περιλαμβάνει την επέκταση του τρόπου χειρισμού των αποτελεσμάτων των ερωτημάτων. Στο πλαίσιο

της εργασίας, προστίθενται δύο επεκτάσεις στην αρχική εκτέλεση ερωτήματος. Η πρώτη αφορά του αρχικού ερωτήματος με βοηθητικά ερωτήματα, ονομαζόμενα (sibling), τα οποία παρέχουν αποτελέσματα παρόμοιων υποσυνόλων του χώρου δεδομένων με του αρχικού. Έτσι, για παράδειγμα, εάν το αρχικό ερώτημα αφορά ένα ατομικό φίλτρο της μορφής «πόλη = Ιωάννινα», θέλουμε να δημιουργήσουμε αυτόματα ερωτήματα sibling για όλες τις πόλεις που αφορούν την ίδια χώρα με τα Ιωάννινα, δηλαδή την Ελλάδα, και να αντιπαραβάλουμε τα αποτελέσματα τους. Η δεύτερη επέκταση, αφορά την εφαρμογή μοντέλων στα δεδομένα των αποτελεσμάτων. Υποθέτοντας ότι μία από τις διαστάσεις ομαδοποίησης είναι ο χρόνος, δημιουργώντας έτσι μια χρονοσειρά ως αποτέλεσμα του ερωτήματος, μοντέλα όπως trend, unimodality, bimodality, or dominance, μπορούν να εφαρμόζονται σε διαφορετικές παραγόμενες χρονοσειρές για τα διάφορα ερωτήματα και συγκρίνοντας τα μεταξύ τους για ομοιότητες και διαφορές. Τα εξαγόμενα highlight βαθμολογούνται, ταξινομούνται και φιλτράρονται με ένα φίλτρο rank-n-prune.
Το αποτέλεσμα ολόκληρης της διαδικασίας είναι (α) ένα σύνολο γραφημάτων που απεικονίζουν τα αποτελέσματα, αλλά το πιο σημαντικό, (β) μια συνδυασμένη αναφορά ή ιστορία δεδομένων, που συνδυάζει τις γραφικές παραστάσεις, μαζί με μια περιγραφή κειμένου των κυριότερων ανιχνευόμενων σημείων, για να επιστραφεί στον αναλυτή.

CHAPTER 1

INTRODUCTION

In the first section of this chapter, we present a brief description of our work and refer to the main directions and the main purpose of our research. In the second section of this chapter, we refer to the structure of this Thesis.

1.1 Goals

The advancement of technology and computer science has facilitated processes that were once laborious and time-consuming. The evolution of databases and especially the optimization of Relational Database Management Systems (RDBMS), played a significant role in the advancement of computing technology. Over time, RDBMSs have become established in a plethora of environments, notably in the business sector.

The increasing popularity of RDBMS in the business world has also created a demand for familiarity with data handling tools. Data handling tools allow users to query and retrieve data from an RDBMS without requiring the user to be proficient in database concepts.

However, their advantage evolved over time into their disadvantage. The reason for this, is that the need arose for further analysis of the data and not simply for retrieving the data. In the daily life of businesses, it is important to retrieve data, analyze it, make decisions and query new or old data again and again.

This endless business cycle of data retrieval-analysis-and-finding needs to be automated and done as fast as possible. Furthermore, the available data is increased daily making the process even more difficult. All these requirements have been a challenge for the computer science sector in recent decades. What emerged through the research was that the traditional database management systems and data handling tools are not capable of solving all the above issues and that it is necessary to evolve these systems into Business Intelligence systems, which process multidimensional data located in data warehouses.

This master thesis provides the extension of a Business Intelligence system with a new subsystem with which an analyst will be able to register a query that interests him and receive graphical visualizations of the query that was performed, and for auxiliary queries. Also, the system automatically passes the results of these queries through extraction algorithms to extract highlights and produces a report with rankings and comparisons between the results of the original query and the auxiliary, and finally produces a report (data story).

Initially, the analyst completes the required text inputs from the GUI (cubeName, groupers, selection filters, aggregation function and a measure) and selects his desired graph visualization type {line-chart, bar-chart and scatterplot}. Supposing the analyst has selected for grouper1 (x-axis) a valid time dimension, thus producing a timeseries from the query results, the system responds with graph visualizations of the original chart and the auxiliary queries, and if the results are of a unique time series, it displays two extra windows, one containing the table with rankings of models for the queries along with the comparisons (unique, common or nothing) between original and auxiliary queries and one info table containing the name of query, the type (basic or sibling) and the sql expression that corresponds to. Finally, and most importantly, the system automatically produces an html report (data story) containing images of charts along with a textual description of the detected and most important highlights extracted from models.

The main contributions of this Thesis are as follows:

- We propose a new querying operator, the *Query-As-A-Chart* operator, that takes a chart specification (line-chart, bar-chart, or scatter plot) as its input and produces a data story as its output.
- The operator includes the extension of the traditional single-query execution, with a palette of related, automatically generated auxiliary queries that will produce the data story. We utilize sibling queries that exploit selection filters of the original query and automatically generate queries with similar selection filters, in order to contrast the results of the original query to the results of its similar siblings.
- The results of the queries are passed through models, like, for example, trend, unimodality, bimodality, or dominance. Then, the models of the different queries can be compared for commonalities and exceptions
- The query result is ultimately a data story, consisting of a fully automatically generated page with graphical representations and textual descriptions of the detected interesting findings.

1.2 Thesis Structure

This Thesis consists of 5 sections. Its structure is as follows: In Section 2, we present the necessary related work and the background of this Thesis. In Section 3, the implementation details for the generation of graphs and extraction algorithms and extraction of report is described along with the tests that were planned. In Section 4, we experimentally evaluate the proposed method, and report on our methodology and the results of the experiments, as well as to the conclusions drawn from them. In Section 5, we discuss the conclusions and the results of this Thesis and we refer the reader to potential future work.

CHAPTER 2

RELATED WORK

In this chapter, we review fundamental concepts that are necessary for the better understanding of our work and related work. Moreover, we make a quick overview of recent related work in the area of data exploration and knowledge discovery.

2.1 Fundamental Concepts

2.1.1 Multi-Dimensional Data

In the context of databases, "multi-dimensional data" refers to data that can be organized and analyzed across multiple dimensions or attributes simultaneously. Briefly, multidimensional data is classified into two main types: (a) facts with associated numerical measures and (b) dimensions that characterize the facts and are more textual [JePT10]. Dimensions are used to group or filter records and the values of dimensions are either categorical (e.g., "City") or temporal (e.g., "Month"). Measures are numerical columns (e.g., "Sales") where certain operations of aggregation (e.g., SUM) can be performed [MDHZ21].
Data models that are used to represent and process multidimensional data are characterized Multidimensional. Multidimensional models have many important application areas within data analysis. The 3 most significant application areas are data warehousing, OLAP systems and data mining [JePT10].

2.1.2 OLAP Systems

The term OLAP abbreviates On-Line Analytical Processing [JePT10]. OLAP systems are systems that facilitate complex analysis of large volume of multi-dimensional data. Furthermore, the analysis in these systems, occurs "On-Line", i.e., fast, "interactive" query response is implied [JePT10]. OLAP systems come in three broad categories: ROLAP systems, MOLAP systems, and HOLAP systems [JePT10].
ROLAP systems store data in traditional relational databases and generate SQL queries on the fly to retrieve data. They provide flexibility and can handle larger datasets but might have slower query performance compared to MOLAP systems. On the other hand, MOLAP systems store data in a multidimensional array-type representation, providing fast query performance due to pre-aggregated data and efficient indexing. Finally, HOLAP systems (hybrid systems) combine the technologies of both MOLAP and ROLAP, allowing users to leverage the advantages of both approaches [Mbaa21].

2.1.3 OLAP Cube

A cube generalizes the tabular spreadsheet such that there can be any number of dimensions and not only two as in spreadsheets [JePT10]. Although instinctively one would think that a cube can only have 3 dimensions, a cube can have any number of dimensions. In this context, dimensions are used for two purposes: the selection of data and the grouping of data at a desired level of detail. Furthermore, each dimension is organized into a containment-like hierarchy composed of a number of levels, each of which represents a level of detail that is of interest to analyses to be performed [JePT10]. For that reason, the term hypercube is sometimes used instead of cube [JePT10], underscoring the multidimensional and hierarchical nature of data representation for advanced analytical purposes.
A cube consists of uniquely identifiable cells at each of the dimensions' intersections. A non-empty cell is called a fact [JePT10]. Typically, a fact derives from the combination of two or more-dimension values. Facts usually have associated numerical properties that are called measures [JePT10]. A measure consists of two parts: a numeric property linked to a fact and a formula for aggregating multiple numeric

values into one. For example, in Figure 2.1, there is a cell that expresses the fact that the product juice was sold in quantity of 10 million for the north region of a country and for month January. In this example, measure is the attribute of total sales, while dimensions are the Region, Time, and Month. The type of product “Juice”, “the North Region”, and the “January Month” are instances of dimensions and the value of 10 million is the measure quantity.
However, a cell may also be empty, without any association with a measure, meaning that there is no information to record for the given combination. Depending on the percentage of the facts (non-empty cells) in cube, cubes are classified in two main categories: dense cubes (higher percentage of cells that are facts) and sparse cubes (lower percentage of facts). In general, increasing dimensions and determining finer granularity of dimension values leads to sparse cubes.
Although there is no theoretical limit to the number of dimensions, typically most cubes in real-world scenarios can have 4-12 dimensions [JePT10, KiRo02, Dyre96]. Generally, only 2 or 3 of the dimensions can be viewed at the same time and that happens due to the inability of the human eye to perceive things larger in more than three dimensional axes. Thus, the dimensionality of a cube is reduced at query time by projecting it down to 2 or 3 dimensions and aggregating of the measure values across the projected-out dimensions [JePT10].
For example, in Figure 2.2, we depict an OLAP cube for loans and orders, that is illustrated through MySQL database-schema. Tables: account, payment_reason, date and status are the dimensions of the OLAP cube, while tables orders and loans are fact-tables. Fact-tables can have “independent” attributes (without any association with other dimensions); for example, in the case of table orders: the “bank_to”, “account_to” and “amount” are independent, and simultaneously an order is a fact because is characterized by the intersection of “order_id” (primary key associated with order dimension), an “account_id” (foreign key associated with account dimension) and a “reason_id” (foreign key associated with payment_reason). Moreover, tables: account, payment_reason, date and status have the “All” attribute, which is an aggregation of their measure values, used in query time. An example of how a fact-table (orders) in tabular form(attributes) and data records (rows) is presented in the Figure 2.3.

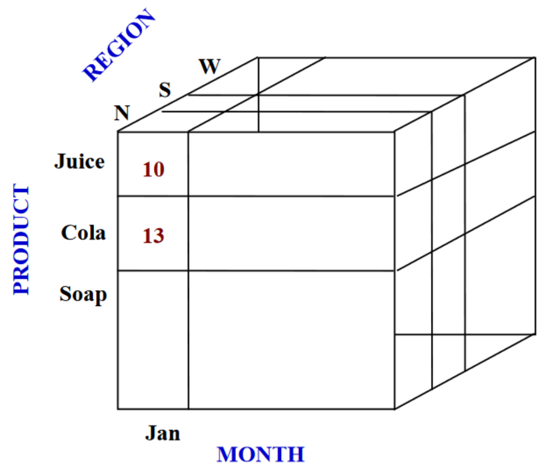


Figure 2.1 A three-dimensional cube (dimensions: Region, Month, Product)

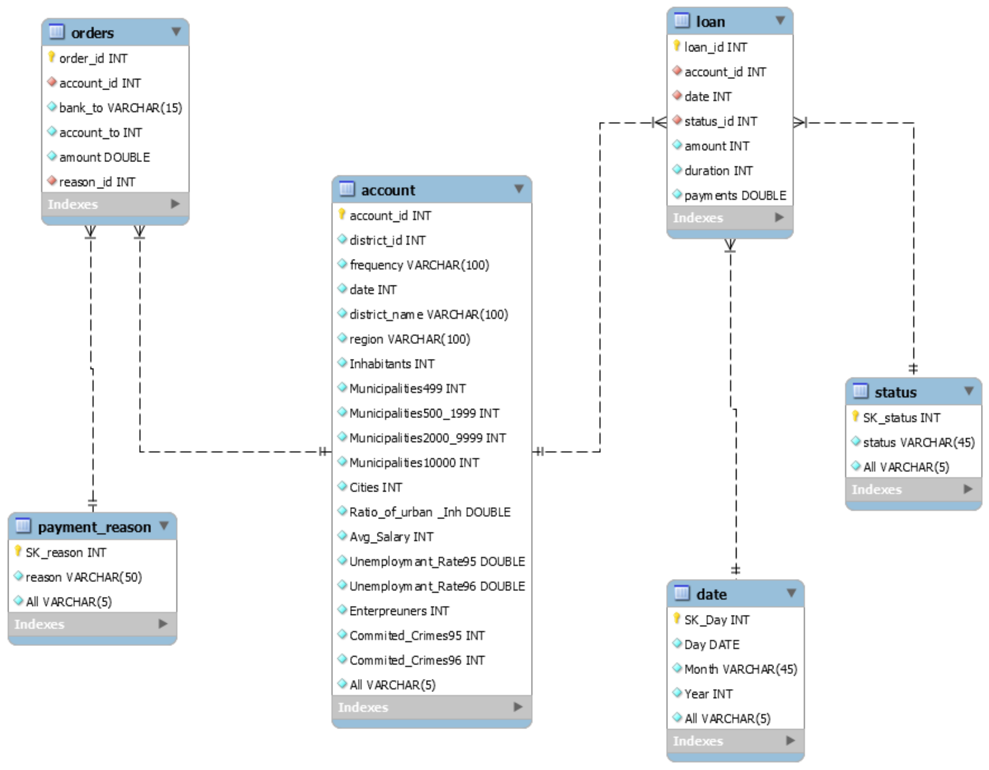
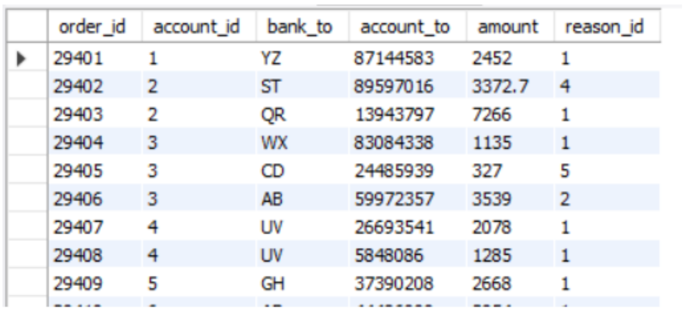


Figure 2.2 A database-schema for an OLAP cube.



order_id	account_id	bank_to	account_to	amount	reason_id
29401	1	YZ	87144583	2452	1
29402	2	ST	89597016	3372.7	4
29403	2	QR	13943797	7266	1
29404	3	WX	83084338	1135	1
29405	3	CD	24485939	327	5
29406	3	AB	59972357	3539	2
29407	4	UV	26693541	2078	1
29408	4	UV	5848086	1285	1
29409	5	GH	37390208	2668	1

Figure 2.3 Fact-table with some records.

2.1.4 EDA

EDA stands for Exploratory Data Analysis, and is an essential step in the data analysis process that involves examining and understanding the data to uncover patterns, relationships, and insights before proceeding to formal modeling or hypothesis testing. EDA is primarily used in data analytics to gain a better understanding of the dataset and to make informed decisions about data preprocessing and modeling strategies. EDA is a tedious task, but it has attracted a lot of attention lately due to its importance.
The current view on interactive visual data analysis has been primarily shaped by John Tukey's emphasis on EDA, who considered data analysis in two stages: exploratory analysis and confirmatory analysis [GHG+22]. Exploratory analysis is the initial stage in which the primary goal is to understand the data, identify patterns, detect anomalies, and generate hypotheses. Confirmatory analysis is the subsequent stage that involves testing specific hypotheses based on prior knowledge or theories performed in exploratory analysis stage.
Contemporary useful tools in the EDA are the EDA notebooks, interactive notebooks that allow the visual and statistical exploration of datasets using deep learning which supposes having access to lots of former analysis or pre-analyzing datasets for computing so-called insights.

2.1.5 Highlights

There is no clear and typical definition for the term of highlights. However, in general we use the term to refer to the significant and noteworthy insights or results

that emerge during the process of examining and interpreting data. For example, in the study [MDHZ21], the authors define highlights and they use the term MetaInsights. In general, examples of significant and noteworthy insights can be various patterns, trends, correlations, anomalies, or other statistical metrics that stand out and are of particular importance. Graphically, these insights can be identified, for example, as spikes or dips in data patterns, clusters of similar data points. Highlights often revolve around KPIs (Key Performance Indicators) that provide a snapshot of the overall health or success of a business or project [Twin23].
For example, a Sales Manager is analyzing the sales data for retail company and notices that during a Black Friday sale event, the company experienced a 30% increase in online sales compared to the previous year's Black Friday, resulting in record-breaking revenue. Sales Manager wants to measure the online sales growth and for this reason defines the following KPI:

- $(\text{Annual Online Sales Growth}) = \frac{(\text{Total Online Sales for Current Year} - \text{Total Online Sales for Previous Year})}{\text{Total Online Sales for Previous Year}}$

Moreover, Sales Manager sets the KPI to achieve a minimum of target to 20% annual online sales growth. Achieving this KPI would be a critical performance measure for the company.
In this example, the highlight is the significant increase in online sales during the Black Friday sale event compared to the previous year, graphically identified as an upward trend. The KPI: "Annual Online Sales Growth," aggregates Online Sales for each day of the year, including Black Friday's event, which serves as a highlight.

2.1.6 Interestingness of Highlights

The "interestingness of highlights" refers to the degree to which certain highlights in data capture the attention of audience. In other words, it is the process of characterizing the highlights meaningful. Often, this characterization takes the form of interestingness scores [MaPV19] for retrieved data or patterns [GeHa06, Bie13]. Typically, a high interestingness score for a finding, expresses how interesting or important it is compared to other findings. Therefore, in most cases, the findings are classified by their interesting scores, rated by a human, system, or data metrics [MaPV19].

2.1.7 KPIs

KPI stands for Key Performance Indicator, a quantifiable measure of performance over time for a specific objective [Twin23]. Business users often make decisions to achieve KPIs such as increasing customer retention or sales or decreasing costs. From finance and HR to marketing and sales, key performance indicators help every area of the business move forward at the strategic level.

Although it may at first sight appear that a KPI is just a metric, they are not the same. KPIs are the key targets that business users should track to make the most impact on their strategic business outcomes [Ruso23]. KPIs support business strategy and help business teams focus on what's important. An example of a KPI is, "targeted new customers per month". On the other hand, Metrics measure the success of everyday business activities that support KPIs. While they impact the outcomes, they're not the most critical measures. For example, a metric is monthly store visits. Analysis Services provide a framework for defining KPIs is provided, exploiting the business data stored in cubes [VaZi14]. Each KPI uses a predefined set of properties. The predefined set consists of 5 properties, which are MDX expressions (query language used to retrieve and manipulate data from multidimensional databases) that return numeric values from a cube. The properties are described next [VaZi14], using as an example a KPI called "Online Sales Performance":

- Value, which returns the actual value of the KPI. Value is a mandatory characteristic of a KPI. For example, the "Value" property of the KPI: "Online Sales Performance", would return the actual value of online sales for a specific period, such as $500,000 in online sales for current month.
- Goal, which returns the goal of the KPI. For example, the "Goal" property would return the target the company has set for online sales during the same period. Let's say the goal for the current month is $550,000 in online sales.
- Status, which returns the status of the KPI. To best represent the value graphically, this expression should return a value between -1 and 1. For example, the "Status" property would return a value that represents how close the company is

to achieving its online sales goal. Status could be calculated as a ratio of actual sales to the goal. If the status value is 0.91, it means the company has achieved 91% of its online sales target.

- Trend, which returns the trend of the KPI over time. As with Status, it should return a value between -1 and 1. The "Trend" property would provide insight into the direction of the online sales performance over time. It could be calculated as the change in online sales compared to the previous period. If the trend value is 0.05, it means online sales have increased by 5% compared to the previous month.
- Weight, which returns the weight of the KPI. If a KPI has a parent KPI, we can define weights to control the contribution of this KPI to its parent. Usually, business enterprises have various KPIs related to the different strategies. If they have a parent KPI that represents overall company performance, they can assign weights to individual KPIs to control their contribution to the parent KPI. For example, if "Online Sales Performance" is just one of several KPIs, its weight could be set to 0.3, indicating that it contributes 30% to the overall company performance KPI.

Overall, KPIs are important for organizations and businesses as they provide a measurable way to assess success, support data-driven decision-making, and drive performance improvement. By establishing specific metrics to evaluate achievements, organizations gain a clear understanding of whether objectives are being met.

2.1.8 Data Storytelling

Data storytelling is a process for communicating information, tailored to a specific audience, with a compelling narration. Data storytelling is the last “ten feet” of the exploratory data analysis and arguably the most important aspect [Yeol20].

Data storytelling merges three key fields of expertise: Data science, Data visualization and Data narration. Data science is the process of collecting, analyzing, and extracting insights from data while Data visualization is the practice of representing data graphically, often using charts, graphs, and visual elements. Data narration is the process of telling stories with insights extracted from data science and data visualization fields. Data narration is an instance of data science where the pipeline focuses on data collection and exploration, answering questions, structuring answers, and finally presenting them to stakeholders [MaPA23]. Data narration is a way of making data and information more engaging, relatable, and understandable for the audience. Instead of just presenting raw data and statistics, data narration uses storytelling elements to bring the data to life and create a meaningful context for the information being shared.

2.2 Related Work on Automatic Discovery of Knowledge and Interactive Learning

Interactive visual data analysis systems have as a primary objective to help business users to make data-driven decisions. However, there are many obstacles that the current analysis process needs to override, especially challenges associated with answering decision leading questions, in order to help business users refine validate hypothesis. The main challenges that the current analysis process face are: the limitation of human working memory and cognitive overload, the difficulty on the scale of interactive exploratory analysis due to increased size and complexity of data, and the fact that the business users may struggle with decision making questions because the suggested solutions may not align with their intuitions.

To address these problems, the authors in [GHG+22] suggest that today's commercial tools must provide four functionalities to enable business users to interactively learn and reason about the relationships (functions) between sets of data attributes thereby facilitating data-driven decision making. These four functionalities are named Driver Importance Analysis, Sensitivity Analysis, Goal Inversion Analysis and Constrained Analysis.

Driver Importance Analysis enables users to implicitly learn functions (models) allowing them to understand the relationships between drivers (input) and KPIs (output) along with the artifacts of these learned relationships. For example, let's say a Sales Manager (business user) cares about the Deal Closing rate (KPI) and has in disposal a dataset with data performed in the past six months, containing columns with information about activities such as making calls, starting chats, attending meetings, opening marketing emails etc. from previous client-users. The forementioned column-activities are the drivers (input), while there is a column named: Deal

Closed? containing a Boolean value (yes or no), from which KPI (output) is derived. Using the Driver Importance Analysis, the Sales Manager can implicitly learn relationships between the drivers (activities) and the KPI (Deal closed), such as what are the top three drivers of deal closing goal. The Driver Importance Analysis is performed by training machine learning models (from Scikit-learn library) to predict KPI values. For continuous KPIs, the authors use linear regression models and for discrete KPIs random forest classifiers. For the linear-regression coefficients and for the random-forest feature importances, the authors choose the driver importance values. To ensure that the model's coefficients are not misleading, the authors verify the importances using traditional measures such as Shapley, Pearson, and Spearman rank. The importance values range between -1 and 1 with extremes showing high negative and positive importance to the KPI respectively while closer to 0 shows low importance to the KPI.

The second functionality, named Sensitivity Analysis, enables users to dynamically evaluate relationships for arbitrary driver values and observe the changes in KPI values. Sensitivity Analysis is performed by making perturbations in the original dataset, such as setting absolute or percentage perturbation magnitudes, and then, perturbated KPI is compared with the original KPI (KPI without perturbations). For example, in continuation with the deal closing rate KPI forementioned scenario, the Sales Manager found from driver importance analysis that the 3 important prospect activities are opening marketing emails, renewing contracts and making calls. Now, the Sales Manager can perform changes in these prospective activities and observe the effects on deal closing rate. He observes that by increasing 100% the rate of open marketing emails, the deal closing rate raises by 4.05%, while by increasing 300% the rate of sending marketing emails, the deal closing rate raises by 5.35% (+1.3%). Therefore, the company will benefit from encouraging activities that lead prospects to open marketing emails instead of sending marketing emails.

The third functionality, named Goal Inversion Analysis, enables the users to interactively set goals such as specific target values or optimization objectives (maximization or minimization) for the KPIs and observe multiple scenarios on how the driver values need to change to achieve the desired goals. Goal Inversion Analysis is performed by setting a desired KPI value (maximum, minimum, or target) and running Bayesian optimizer model (using Scikit-Optimizer's Bayesian optimizer) to receive

the driver values that will achieve the desired-specified KPI. For example, the Sales Manager observes and sets multiple times what prospect activities will increase the deal closing rate by 75%? So, optimizing the KPI ‘s driver to a 75% target gives 78.38 deal closing rate with high confidence and simultaneously receives the values of prospect activities that will enable this goal to be achieved.
The fourth functionality, named Constrained Analysis, allows a user to interactively set conditions over how the learned functions (models) are evaluated or inverted, enabling users to incorporate their domain knowledge such as business constraints and common sense to regulate these functions. For example, consider that the sales manager is under a budget and cannot invest in increasing all the activity values to optimal ones as proposed by Goal Inversion analysis functionality. So, the Sales Manager sets constraints to the minimum and/or maximum value, constraining the range value of some prospect activities, (let’s say: 5%-10% increasing on open marketing emails) and perform goal inversion analysis that satisfy these constraints. On specifying these constraints, and performing goal inversion analysis, the Sales Manager finds that the maximal deal closing rate that can be achieved is about 46% which is still an uplift of about 4 from the original deal closing rate and is feasible for the company.
To this end, in the study [GHG+22], the authors created SYSTEMD, an interactive visual data analysis system enabling business users to experiment with the data by asking what-if questions (performing the forementioned functionalities). The authors evaluated the system through three business use cases: marketing mix modeling, customer retention analysis, and deal closing analysis, and report on feedback from multiple business users. Overall, business users found the SYSTEMD functionalities highly useful for quick testing and validation of their hypotheses around their KPIs of interest, clearly addressing their unmet analysis needs.

Exploratory Data Analysis (EDA) emphasizes in gaining knowledge of data and is a primary step in facilitating further in-depth analysis. In recent years, automatic EDA, which focuses on automatically discovering pieces of knowledge in the form of interesting data patterns, has been an emerging topic.
However, the knowledge conveyed by the suggested data patterns of EDA is disjointed or lacks organization. Therefore, it is difficult for users to gain structured

knowledge (i.e., knowledge of how facts or concepts are organized by certain relations). As the number of suggested patterns grows, these stand-alone patterns are likely to motivate users to conduct follow-up analysis. This in turn hinders the suggested patterns being effectively utilized to facilitate EDA.
In the study [MDHZ21], the authors propose a structured representation of knowledge extracted from multidimensional data, named MetaInsight, which aims to facilitate EDA effectively. Specifically, the authors propose a novel formulation of basic data patterns to capture essential characteristics of raw data distribution for knowledge extraction. For example, the "Outstand" data pattern is illustrated as a subspace with the highest aggregate values, "Trend" as an upward or downward trend, "Outlier" as position of outlier data points, "Seasonality" as length of seasonality period and "Unimodality" as position of extreme point of a U-shaped valley or peak shape, etc. Furthermore, because of the multiple data patterns that exist in data science field, the authors provide in their proposed work the capability of addition of custom-user data patterns in the basic data patterns.
For better understanding of their work, the authors in [MDHZ21] provide three basic definitions for what is Subspace, Sibling group and Breakdown. The authors define a subspace $s = \{s_1, ..., s_d\}$ as a size-d set of filters on each dimension of the multidimensional dataset, where $s_i \in \text{dom}(\text{col}_i) \cup \{*\}$, where $s_i = \{*\}$ refers to "any" value in dimension with index i of s (i.e., empty filter), and $\text{dom}(\text{col}_i)$ refers to available values for column with index i. For example, a subspace can be {City: Los Angeles, House Style: *, Month: *}. As a Sibling group, the authors define the subspaces that differ from each other in one non-empty filter, for example {City: Los Angeles, April} and {City: Yuba, April}. Last, the authors define a Breakdown dimension as the dimension where group by operator is performed. The Result of the application of a breakdown dimension in a subspace is the generation of a sibling group. For example, when we break down {Los Angeles, Month: *} by dimension "Month", we obtain a sibling group like: {{Los Angeles, Jan}, {{Los Angeles, Feb}, ...}.
Using the three above definitions, the authors define a Data Scope, ds, as a tuple containing a subspace, a breakdown dimension, and a measure (= a numerical dimension of the dataset where an aggregation operator can be performed). To facilitate the data pattern generation of the MetaInsight Algorithm, first, a user must provide a Data Scope from the dataset. For example, ds: {{Los Angeles, Month: *,

..., Sales: *}, Month, SUM(Sales)}, is a data scope with subspace = {Los Angeles, Month: *, ..., Sales: *}, breakdown dimension = Month and a measure= SUM(Sales). Then, based on the user's determined data scope, a query is constructed internally. For the above example: "SELECT Month, SUM(Sales) FROM DATASET WHERE City= "Los Angeles" GROUP BY Month" is the constructed query. After the execution of the query, a sibling group is generated automatically. Then, follows the identification of basic predefined data patterns, constrained by the type of breakdown dimension (for example for Month: Temporal), and a highlight or highlights are generated. For the above example, let's say the type of highlight that was discovered is "Unimodality" and the point that was observed the extreme point is the month "April". The forementioned result can be interpreted in natural language as "Los Angeles has minimum Sales in April".

After the discovery of a highlight/basic data pattern for a given data scope, the next step is the creation of Homogeneous Data Scope (HDS). An HDS for a given data scope, is a set of data scopes and is generated through Subspace Extending or Measure Extending or Breakdown Extending strategy. For example, for the previous scenario: "Sales in Los Angeles", with Subspace Extending we obtain data scopes that correspond to sales in different cities over months. With Measure Extending, we have a set of data scopes with different measures (e.g., SUM(Sales), SUM(Profits), AVG (Profit Rate)) in Los Angeles over Months. Finally, with Breakdown Extending, we obtain a set of time series of sales in Los Angeles with different granularities (e.g., "Day", "Week", "Month").

After the generation of the Homogeneous Data Scope HDS, the next steps are the generation of Homogenous Data Pattern within HDS and furthermore the categorizing of basic data patterns into commonness(es) and exceptions. Homogenous Data Pattern is a set of type-induced data patterns derived from an HDS. For example, let's say by Subspace Extending, we obtain data patterns that are identical (Month and SUM(Sales)) and only differ in the subspace (different city), so they form and HDS but in addition to this fact, it seems from the charts, that all data patterns within HDS also contain the same generated type of Highlight that is Unimodality. So finally, the form an HDP. After that, data patterns within the same HDP are categorized into commonness sets, and exception set. Two data patterns or more, belong to the same commonness set if they have the exact same highlight (meaning they

have the same highlight type and the same value position of the highlight). For example, let's say that three cities of California, have minimum SUM(Sales) in April, while two other cities of California, have minimum SUM(Sales) in October. Then, we have two commonness sets, one with cardinality 3 and the other with cardinality 2. Exceptions denote the data patterns that do not belong to any commonness set. For example, we have only one city with minimum SUM(Sales) in July, then this data pattern belongs to the exception set.
Finally, for the definition of a MetaInsight within the same HDP, the authors provide the extracted commonness sets and the extracted exceptions set. This way, the MetaInsight concretizes knowledge obtained by induction and validation processes which are typically performed in EDA.
Furthermore, the authors in order to automatically discover high-quality MetaInsights, propose a novel scoring function to quantify the usefulness of MetaInsights, as an effective mining procedure and a ranking algorithm. The evaluation of their proposed work was conducted on both real-world datasets and user studies, demonstrating the effectiveness and efficiency of MetaInsight in facilitating EDA.

CHAPTER 3

DATA STORIES FOR CHART QUERIES

In this chapter, we first address the problem definition and resolution by explaining the purpose of the DelianCubeEngine software, the necessary programming tools for its deployment, and presenting our new method overview. Next, we describe step-by-step process for Chart-Query execution. Finally, we describe planned validation tests for verifying the system's performance and accuracy.

3.1 Problem Definition and Method Overview

3.1.1 The Delian Cubes Query Engine

DelianCubeEngine [DeCe18] is an OLAP query answering system. More specifically, the system provides a connection to a database, receives as input specially formatted queries designed for the system, cube queries, and returns the query results in tab-delimited text files.

The format of the input queries, can be either conventional query (queries defined with *CubeName*, *Name*, *Aggrfunc*, *Measure*, *Gamma* and *Sigma*, standard words) or natural language query (queries expressed in a language that resembles the human natural language, i.e., "*Describe the avg of loan amount per account_dim.district_name and date_dim.month for account_dim.region is 'north Moravia' as LoanQuery11_S1_CG-Prtl*").

In addition to the various forms that the system can receive as input query, it can furthermore process the query results, producing statistical models (model package), and furthermore analyze the query itself, producing additional relative queries that may offer important insights for the analysis of the input data (analyze package). We detail cube queries and models in Section 3.1.4 and the system architecture in the subsequent sections.

3.1.2 Problem Definition

The goal of this Thesis is to explore the possibility of (a) defining a query as a chart to be constructed, and (b) to enrich the result of the query with (b1) results of auxiliary queries and (b2) highlights from model extraction algorithms, applied over the query results. The entire result is wrapped as a data story composed of text and graphical representations summarizing the phenomena that have been detected via the model extraction algorithms.

A side effect of the effort is the addition of the graphical visualization via charts for the results of the input query and for the results of the additional relative queries produced in the DelianCubes query engine.

3.1.3 Method Overview

The main steps of the solution of the problem are the following:

1. First, the analyst specifies the intended cube query as a chart, which can be a bar-chart, line-chart or scatter-plot. This means that practically, the query will have two grouper attributes (dimension levels) that will be visualized in the two axes of the chart, and, consequently an aggregated measure. Filters are also parts of the specification
2. Second, in order to analyze the results of the original query, auxiliary queries are automatically generated, in order to contextualize and assess the results of the original query.
3. The queries are executed by Delian Cubes over the underlying data cube, and their results are produced.
4. The results can be immediately visualized, but most importantly, they are passed through a set of pattern checkers / highlight extractors where they are checked for interesting properties that they might hold. Such highlights

include the existence of trend, unimodality, bimodality in time-series query results (where one grouper is time), the existence of a strong linearity in the results as demonstrated by a strong linear regression score, and others.

5. The highlights of the original query are contrasted to the ones of the auxiliary ones, to identify commonalities or exceptions in them.
6. A story maker combines charts, highlights and the text automatically generated for them in a data story presented to the user.

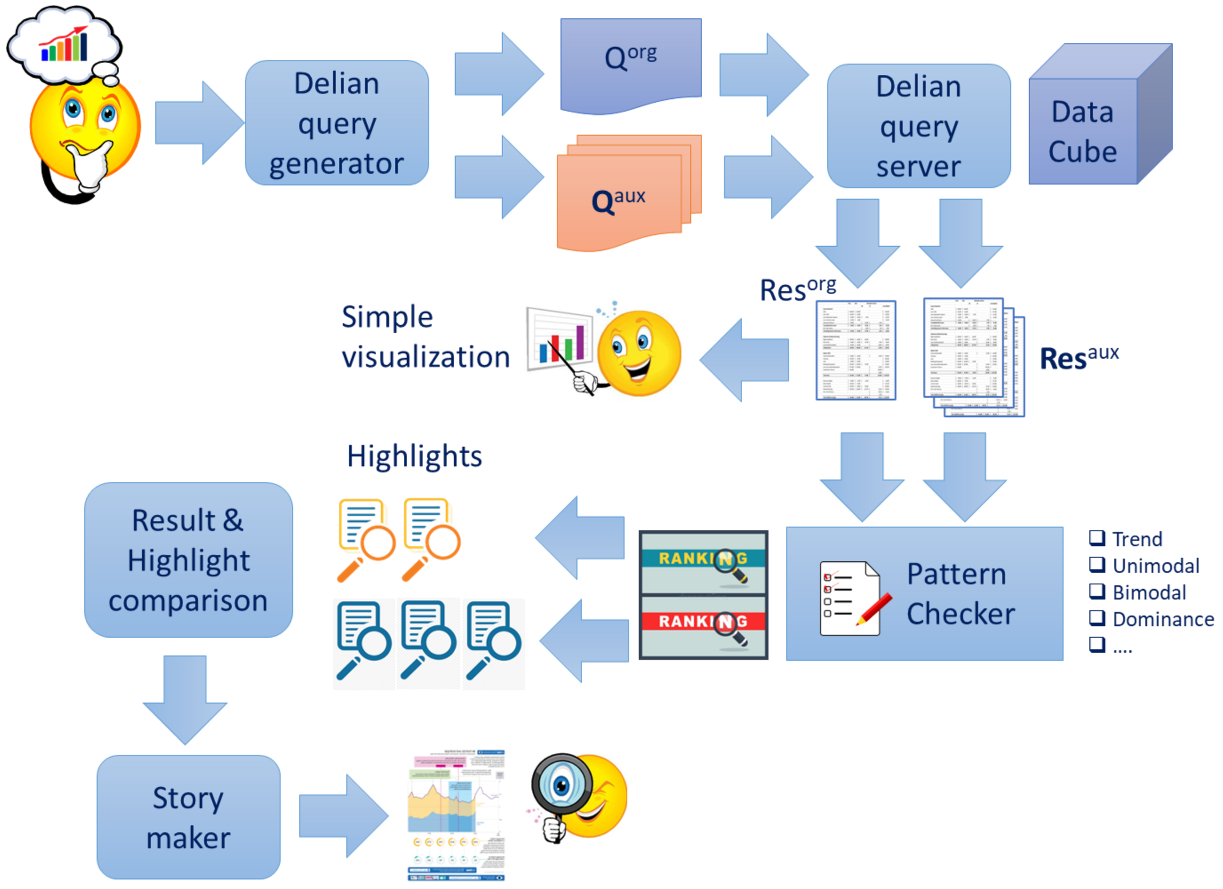
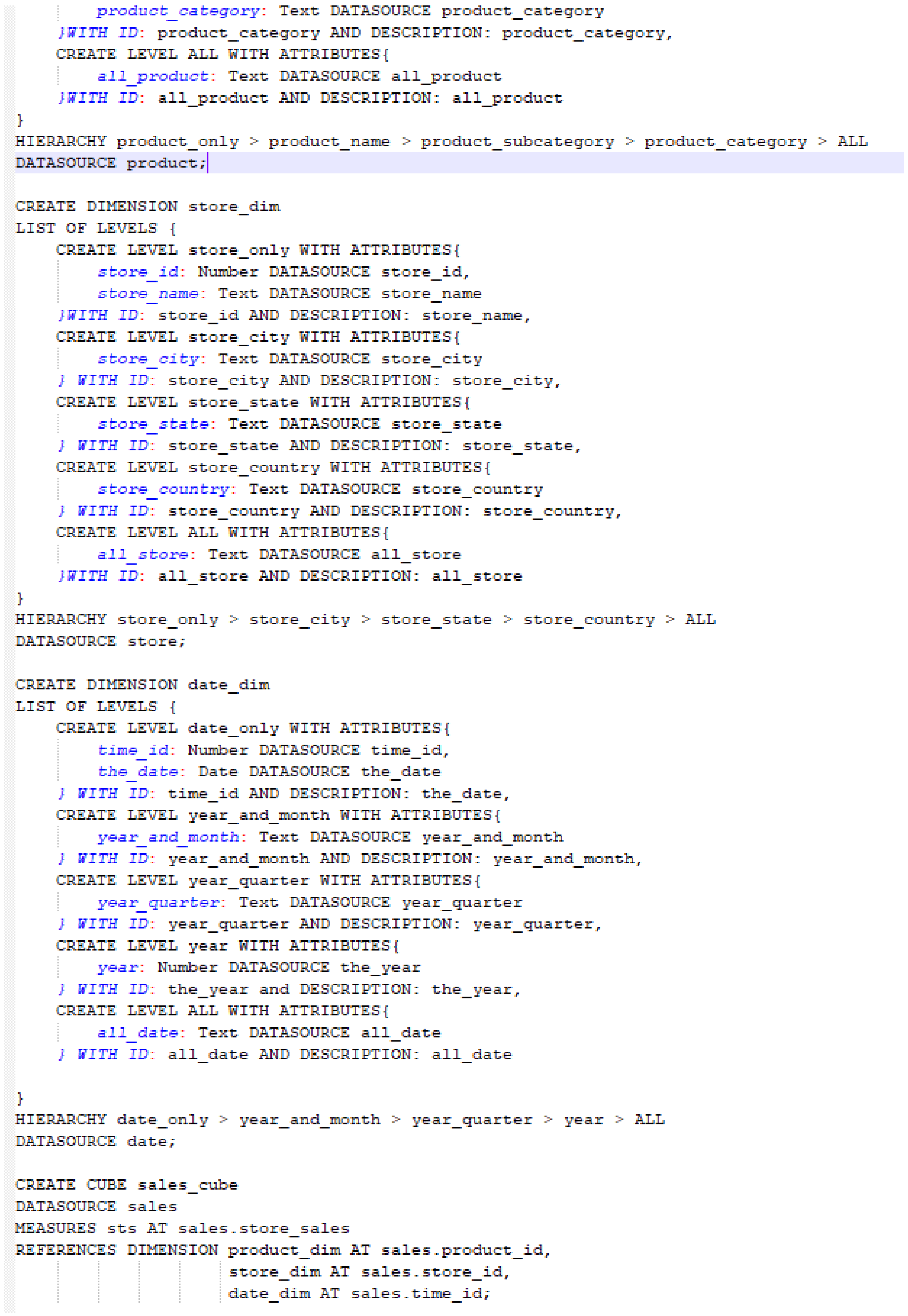


Figure 3.1 Method Overview

3.1.4 Queries and Models

A cube query is a type of query used in OLAP (Online Analytical Processing) systems to retrieve and manipulate data stored in an OLAP data cube.

Before being able to handle queries, a query answering system must have the data cubes registered by the analyst. In our Delian query server, a Data cube can be parsed and defined via input files in the form of .ini file. However, to create a data cube, we need to define the dimensions and the hierarchies of the cube following the Delian's cube grammar and syntax (CubeSqlLexer, CubeSqParser and CubeSql.g). An example of a valid .ini file is shown in Figure 3.2.



```
        product_category: Text DATASOURCE product_category
    }WITH ID: product_category AND DESCRIPTION: product_category,
    CREATE LEVEL ALL WITH ATTRIBUTES{
        all_product: Text DATASOURCE all_product
    }WITH ID: all_product AND DESCRIPTION: all_product
}
HIERARCHY product_only > product_name > product_subcategory > product_category > ALL
DATASOURCE product;

CREATE DIMENSION store_dim
LIST OF LEVELS {
    CREATE LEVEL store_only WITH ATTRIBUTES{
        store_id: Number DATASOURCE store_id,
        store_name: Text DATASOURCE store_name
    }WITH ID: store_id AND DESCRIPTION: store_name,
    CREATE LEVEL store_city WITH ATTRIBUTES{
        store_city: Text DATASOURCE store_city
    } WITH ID: store_city AND DESCRIPTION: store_city,
    CREATE LEVEL store_state WITH ATTRIBUTES{
        store_state: Text DATASOURCE store_state
    } WITH ID: store_state AND DESCRIPTION: store_state,
    CREATE LEVEL store_country WITH ATTRIBUTES{
        store_country: Text DATASOURCE store_country
    } WITH ID: store_country AND DESCRIPTION: store_country,
    CREATE LEVEL ALL WITH ATTRIBUTES{
        all_store: Text DATASOURCE all_store
    }WITH ID: all_store AND DESCRIPTION: all_store
}
HIERARCHY store_only > store_city > store_state > store_country > ALL
DATASOURCE store;

CREATE DIMENSION date_dim
LIST OF LEVELS {
    CREATE LEVEL date_only WITH ATTRIBUTES{
        time_id: Number DATASOURCE time_id,
        the_date: Date DATASOURCE the_date
    } WITH ID: time_id AND DESCRIPTION: the_date,
    CREATE LEVEL year_and_month WITH ATTRIBUTES{
        year_and_month: Text DATASOURCE year_and_month
    } WITH ID: year_and_month AND DESCRIPTION: year_and_month,
    CREATE LEVEL year_quarter WITH ATTRIBUTES{
        year_quarter: Text DATASOURCE year_quarter
    } WITH ID: year_quarter AND DESCRIPTION: year_quarter,
    CREATE LEVEL year WITH ATTRIBUTES{
        year: Number DATASOURCE the_year
    } WITH ID: the_year and DESCRIPTION: the_year,
    CREATE LEVEL ALL WITH ATTRIBUTES{
        all_date: Text DATASOURCE all_date
    } WITH ID: all_date AND DESCRIPTION: all_date

}
HIERARCHY date_only > year_and_month > year_quarter > year > ALL
DATASOURCE date;

CREATE CUBE sales_cube
DATASOURCE sales
MEASURES sts AT sales.store_sales
REFERENCES DIMENSION product_dim AT sales.product_id,
                     store_dim AT sales.store_id,
                     date_dim AT sales.time_id;
```

Figure 3.2 Configuration File for a data cube

A cube query in the Delian cube server is defined as follows:

- CubeName: Name of the cube
- Name: Name for the cube query
- AggrFunc: Aggregation Function ∈ {SUM, COUNT, AVG, MIN, MAX}
- Measure: Measure, a fact column of the cube
- Gamma: Grouper levels -- columns used in GROUP BY SQL query
- Sigma: A conjunction of atomic filters of the form *Level = value*, mapped to the respective atoms in the WHERE clause of an SQL query.

The **notation** that we use for cube queries is as follows

$$q = \gamma_{D1.L1,D2.L2}^{agg(M)}\left(\sigma_{\varphi}(C)\right), \varphi: \bigwedge D_i.L_i = v_i$$

where C is the cube name, σ_φ is the sigma selection condition, M is the measure, agg is the aggregate function applied to it, and γ is the group-by operator with the two grouper levels as subscripts and the aggregated measure as superscript.

The **semantics** of a cube query in SQL are as follows:

SELECT D1.L1, D2.L2, Agg(M)

FROM C

WHERE $\sigma_{\varphi}(C)$

GROUP BY $\gamma_{D1.L1,D2.L2}$

An example of a Delian cube query for the corresponding cube sales_cube:

CubeName: sales

Name: SalesQuery11_S1_CG-Prtl

AggrFunc: Avg

Measure: store_sales

Gamma: date_dim.lvl2, store_dim.lvl2

Sigma: date_dim.lvl3 = ‘1997’, store_dim.lvl3= ‘USA’.

A particularity of our chart-queries extension to Delian Cubes is that once a cube query has been issued, we are interested to find auxiliary queries that contextualize and assess the results of the original query. An auxiliary query is a query that is semantically related to the original query, but comes with some planned mutation that allows to complement the original results with similar data – for example, we can retain the same Measure and aggregation function but change the filter conditions to produce “peer” queries that cover different parts of the data space than the

original one, or/and, to change the groupers in order to attain coarser or more detailed information.
In the context of this master thesis, auxiliary queries are **sibling queries**, that derive from the original with "relaxation" of its filter conditions. First, we formalize the construction of sibling queries. A sibling query is produced in the following way:
For every atomic filter condition of the sigma selection condition say $D.L = v$, we

(a) change the filter to D.parent(L) = parent(v)

(b) replace one of the groupers with D.L

(c) keep the other conditions and the other grouper unchanged.

This practically means that for every atom, two sibling queries are generated, one for each grouper that is replaced with D.L.

We give an example to illustrate the concept. Assume the original query with name SalesQuery11_S1_CG-Prtl. The query has two atomic filter conditions: date_dim.lvl3 = '1997' and store_dim.lvl2 = 'USA'. The hierarchy of the dimension store_dim (Figure 3.2) is defined as:

$$store\_only > store\_city > store\_state > store\_country > ALL.$$

The semantics behind the hierarchy is that a store_only belongs to a store_city, a store_city belongs to a store_state and a store_state belongs to a store_country and a store_country to all.

An example of auxiliary query produced from the original is depicted in the following table. For date_dim, the level year is changed to the parent level that belongs to "ALL" and simultaneously the grouper of date_dim becomes lvl3 = year, that was the filter condition level from the original.

CubeName: sales Name: SalesQuery11_S1_CG-Prtl AggrFunc: Avg Measure: store_sales Gamma: **date_dim.lvl2**, store_dim.lvl2 Sigma: **date_dim.lvl3 = '1997'**, store_dim.lvl3= 'USA'.	CubeName: sales Name: SalesQuery12_S1_CG-Prtl AggrFunc: Avg Measure: store_sales Gamma: **date_dim.lvl3**, store_dim.lvl2 Sigma: **date_dim.lvl4 = 'ALL',** store_dim.lvl3= 'USA'.
Original query	Sibling Query

We consider different **query classes** in our setup. Here we list the four more relevant to this thesis:

- The most general query class concerns **simple cube queries**: Simple cube queries are the most general case of cube queries with any levels of different dimensions as groupers, and no particular relationship of filters with groupers.
- **Time-series cube queries**, also referred to as timeseries: for this query class, the first grouper (obligatorily) concerns a level of a time-related dimension (e.g., month, year, decade etc.). This allows the result to be a timeseries of aggregate measurements for the second grouper (and, thus visualized as a line-chart, with time in the horizontal axis). The relationship of groupers with filters can be arbitrary.

$$q = \gamma_{T.L, D2.L2}^{agg(M)}\left(\sigma_{\varphi}(C)\right), \varphi\text{: } \bigwedge D_i.L_i = v_i$$

- **Single grouper pinned queries**: in this case, the filter includes a selection filter for a grouper. Thus, assuming a grouper of the form $D.L^g$, for dimension D and level L^g, this query class also includes a filter of the form $D.L^s = value$, for the same dimension and a level L^s which is higher or equal to L^g. If L^s is identical to L^g, then the query produces a single value for the respective grouper.
- A time-series where the second grouper has been pinned to a single value is a **Unique Time-series Query**. In other words, the first grouper concerns time, and the second grouper includes a filter at the same level as the grouper, resulting in a single timeseries as the query result.

$$q = \gamma_{T.L, D2.L2}^{agg(M)}\left(\sigma_{\varphi}(C)\right), \varphi \ni D_{2.}L_2 = v$$

Apart from query classes in our setup, we determine mathematical **models** that aim to extract interesting phenomena found in results for both the original and auxiliary queries. We have implemented the following models only for Time-series cube queries presented above with their names in Delian:

- **AbsoluteTrendModel:** Model that finds if timeseries is absolutely monotonically increasing (uptrend) or decreasing (downtrend) or none of them.

- **ContributorModel:** Model that finds if timeseries has a time x-axis value that contributes > 50% **(Mega contributor)** in the produced results.
- **KendallBasedModel:** Model that finds if timeseries is monotonically increasing(uptrend) or decreasing(downtrend) based on the Kendall tau coefficient. In this thesis, we use the Kendall coefficient from Apache Commons.
- **ModalityModel:** Unimodality or Bimodality finder for the timeseries. A timeseries is considered unimodal when it forms a U-shaped valley or peak shape. A timeseries is considered bimodal A when it has two distinct peaks or modes in its distribution
- **RegressionModel:** Model that performs Linear Regression using SimpleRegression provided by Apache Common library.

To determine the importance of results of the model we determined a score function from each, producing a score in the range [-1,1]:

- **AbsoluteTrendModel score:**

$$\text{score} = \begin{cases} -1, & \mathit{downtrend} \\ 0\,, & \mathit{no\ trend} \\ 1\,, & \mathit{uptrend} \end{cases}$$

- **ContributorModel score:** The score is computed as the ratio of the maximum measure of grouper1 values to the sum of measures across all grouper1 instances.
- **KendallBasedModel score:** score = abs(tau) where tau is Kendall's coefficient.
- **ModalityModel score:**
  To compute the score for the timeseries we follow the above steps:
  - Divide the series into segments from the start to each point where the sign of y changes.
  - For each segment, if the initial or final y-values are negative, determine the offset by taking the absolute value of the minimum of the initial and final y-values (score $= \frac{|maxValue| - |minValue|}{|maxValue|}$).
  - We adjust values by adding the offset plus one to both the initial and final y-values to make them positive.
  - Use the adjusted positive values to compute the score for segment using the standard formula applicable for positive values.

- Finally, we compute the score for the timeseries by calculating the average of the scores from all segments:
  $(\frac{1}{N-1}\ \sum_{0}^{N-1} score(segment),\ where\ N\ the\ number\ of\ segments).$

- **RegressionModel score:** score = 1- MSE (Mean Square Error) produced from Linear Regression provided by Simple Regression Apache Common Library.

Furthermore, some models may not offer some important info (highlights) that shouldn't be concluded to the final produced story. For that reason, we filter the highlights according to their score. As it was mentioned before, the score range between [-1,1]. Some negative scores are important (for example downtrend) and for that reason we consider the absolute value of every score. We determined a threshold θ (in our deliberations, θ = 0.5) above which a highlight is important.
The formula for highlight selection is described below:

$$\text{Highlight} = \begin{cases} important\ , & score > \theta \\ unimportant & score \leq \theta \end{cases}$$

3.1.5 Example of Usage

The execution process of our work starts when a user selects from the MainApp window, in the horizontal menu, the option "Work". Then, the user selects the option "Run Chart Query" from the window that appears (Figure 3.3).

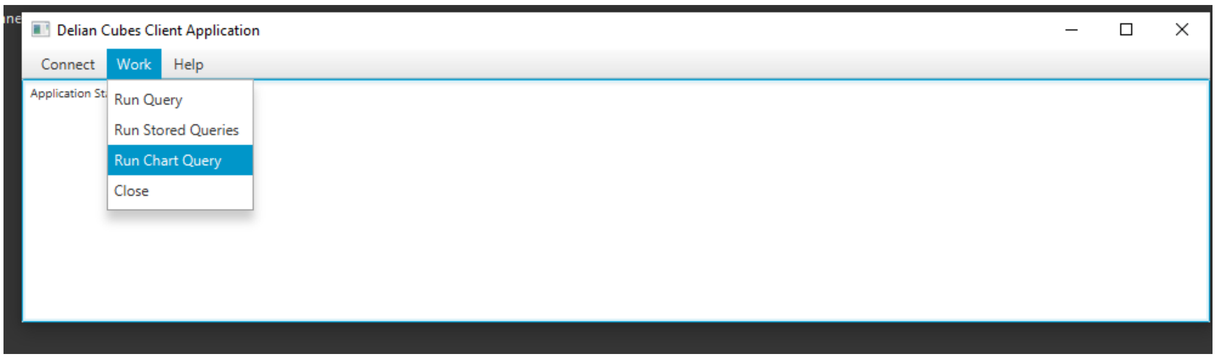


Figure 3.3 Addition of "Run Chart Query" in initial window of DelianCube Application

After clicking the option "Run Chart Query", a new window pops up with name "ChartQueryEditor". The user may enter the necessary text fields (Cube name, Data series Grouper 1, Data series Grouper 2, Measure Column, Filter Column name and Filter Column value), selects Aggregation Function and specify Chart (available options: Bar chart, Scatter Plot and Line Chart) and finally click the button "Run Query" (Figure 3.4).

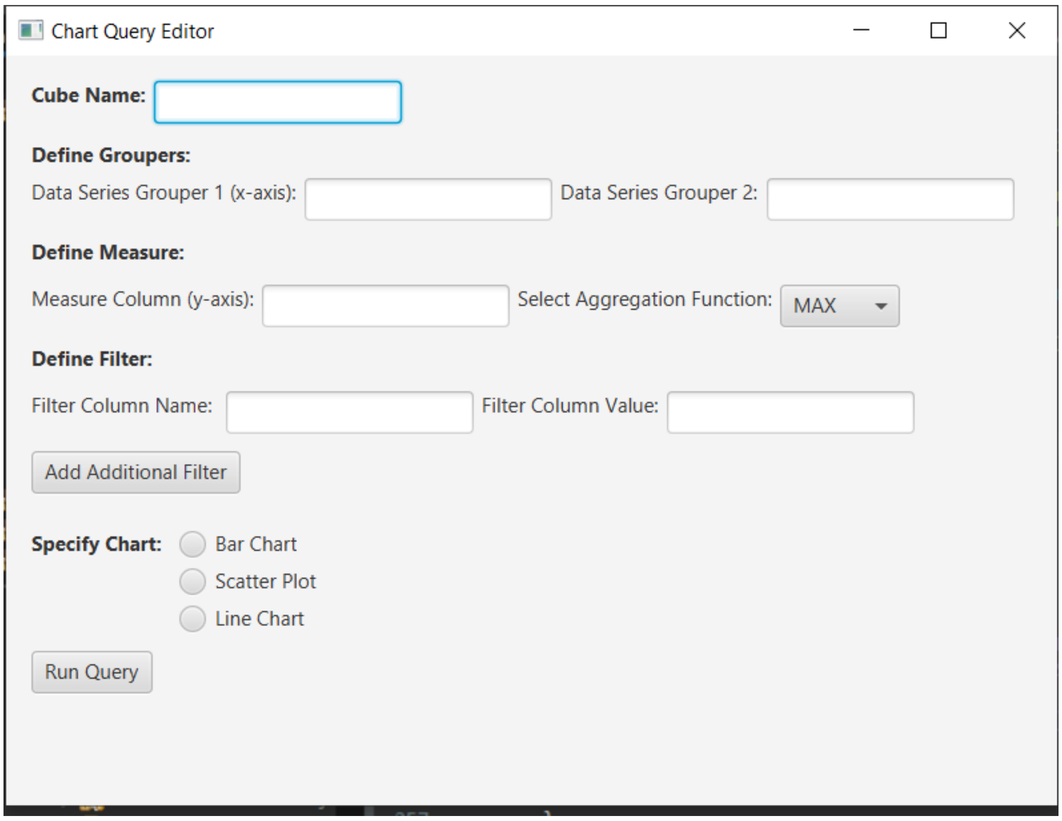


Figure 3.4 Chart Query Editor window

The process of executing the user request is as follows. First, the system converts the user's input (text fields and selected Aggregation Function) to an "analyze query". Then, the analyze query with the selected chart option are packed into a "chart request object" and sent to the back end of the Delian Cubes system.

Subsequently, the chart request object is further processed in order to produce auxiliary queries (specifically sibling queries) via the ANALYZE package. Then, the results of produced queries, are passed through model extraction algorithms to extract highlights from the data. Finally, (a) graphical visualizations of the original and the auxiliary queries are returned to the user, (b) a report (datastory) is produced in the folder OutputFiles of Delian project, containing produced charts along with extracted highlights and if the original query produced a **Unique Time-series Query,** the report contains extra a comparison between the original query 's highlights and its siblings. A very naïve workflow of how timeseries is processed and in the case that is Time-series and in the case that is a unique Time-series is presented in Figure 3.5.

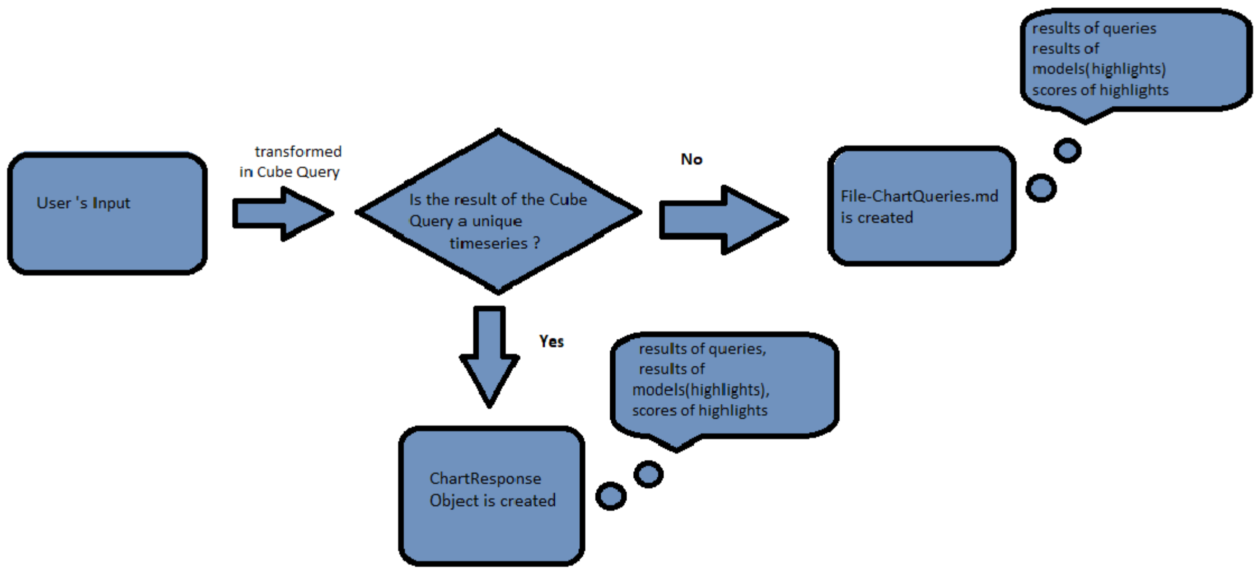

User 's Input
transformed in Cube Query
Is the result of the Cube Query a unique timeseries ?
No
File-ChartQueries.md is created
results of queries results of models(highlights) scores of highlights
Yes
ChartResponse Object is created
results of queries, results of models(highlights), scores of highlights

Figure 3.5 Workflow for handling Timeseries type.

To make the execution process more understandable, the following example is provided [from now on will be referred as example 1]. Consider the following scenario, that the user gives the following input in the ChartQueryEditor window (Figure 3.6):

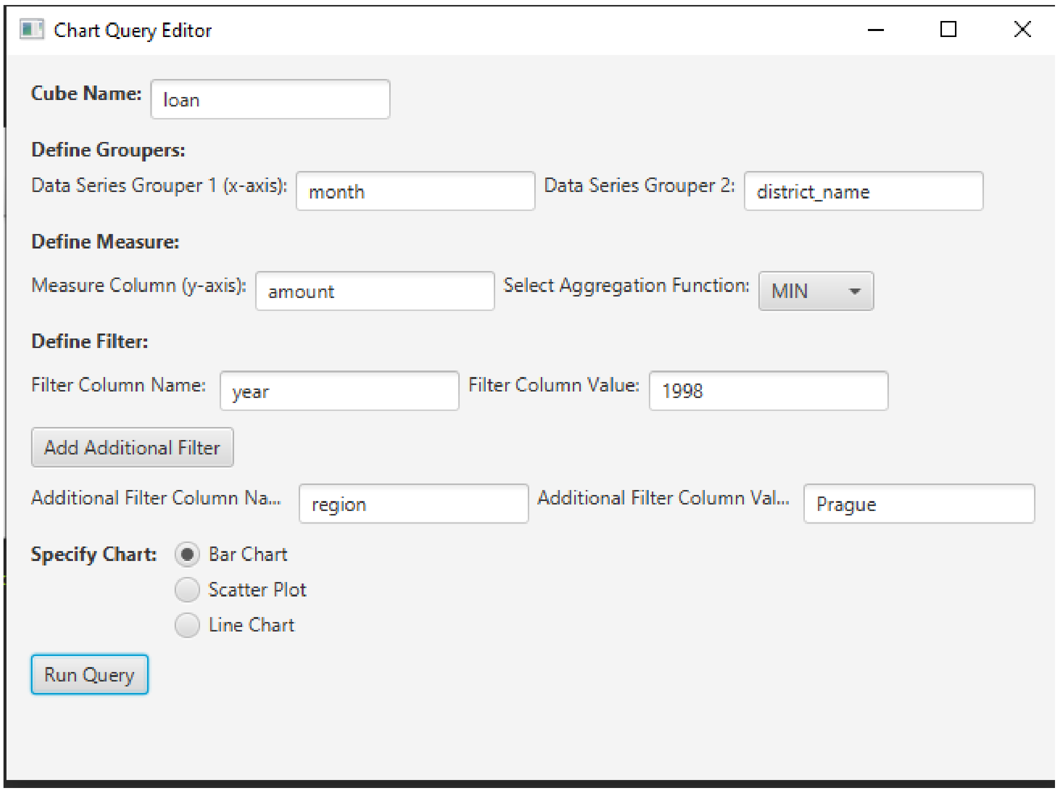


Chart Query Editor

Cube Name: loan

Define Groupers:

Data Series Grouper 1 (x-axis): month   Data Series Grouper 2: district_name

Define Measure:

Measure Column (y-axis): amount   Select Aggregation Function: MIN

Define Filter:

Filter Column Name: year   Filter Column Value: 1998

Add Additional Filter

Additional Filter Column Na... region   Additional Filter Column Val... Prague

Specify Chart: [✓ Bar Chart] Scatter Plot Line Chart

Run Query

Figure 3.6 Example of user Input in ChartQueryEditor window

The user's input is converted to the following analyze query: "ANALYZE MIN (amount) FROM loan FOR year = '1998' AND FOR region = 'Prague' GROUP BY month, district_name AS analyze_query". This is extended with the visualization type via the following chart request object: cr = < "ANALYZE MIN (amount) FROM loan FOR year = '1998' AND FOR region = 'Prague' GROUP BY month, district_name AS analyze_query", "Bar Chart">.

cubeName	Name	aggrFunc	Measure	Gamma	Sigma	Chart
loan	analyze_query-AnalyzeBaseQuery	Min	amount	month, district_name	Year ='1998, region='Prague'	BarChart

Table 3.1.5: Base Cube Query of Example 1

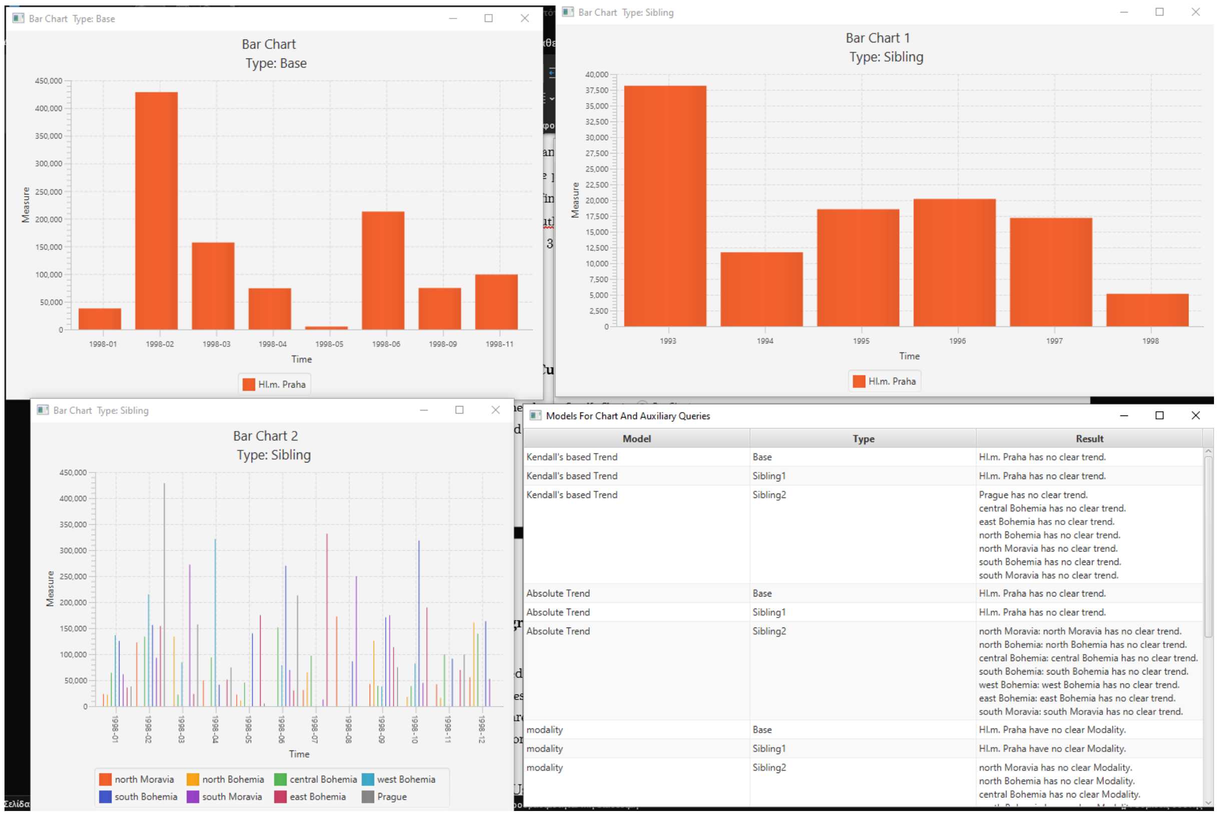
The queries that are produced through the analyze operator (package that already exists in Delian) with the addition of chart specification are the following:

i. The basic cube query resulting from the direct translation.
ii. The sibling queries of the original one

cubeName	Name	ag-grFunc	Meas-ure	Gamma	Sigma	Chart
loan	analyze_query-AnalyzeSiblingQuery_	min	amount	month, region	year ='1998', region='ALL'	Bar-Chart
loan	Analyze_query-AnalyzeSiblingQuery_	min	amount	year, dis-trict_name	Year ='ALL', Region='Pra-gue'	Bar-Chart

Table 3.1.6 Sibling Queries of Example 1.

Afterwards, the results from the execution of the generated Cube Queries, as well as details about their visualization as charts, are saved to a markdown file (.md) in the OutputFiles folder with the name File-ChartQueries_Report.md.
After the creation of the file, we parse the file to take the results of the queries and pass through model extraction algorithms to extract highlights. Then, the highlights are reported to the File-ChartQueries_Report.md (as the result is a multiple series). Finally, the report File-ChartQueries_Report.md is again parsed to create the graph visualizations of the original and the auxiliary queries. Furthermore, a new window with the models that take place and the extracted highlights from each model and query is presented, and finally a report (datastory) with name report.html is created automatically in outputFolder. The pop-up windows for chart queries for example 1 is presented (figure 3.7) and a preview of the report in figure 3.8.



Models For Chart And Auxiliary Queries

Model	Type	Result
Kendall's based Trend	Base	Hl.m. Praha has no clear trend.
Kendall's based Trend	Sibling1	Hl.m. Praha has no clear trend.
Kendall's based Trend	Sibling2	Prague has no clear trend. central Bohemia has no clear trend. east Bohemia has no clear trend. north Bohemia has no clear trend. north Moravia has no clear trend. south Bohemia has no clear trend. south Moravia has no clear trend.
Absolute Trend	Base	Hl.m. Praha has no clear trend.
Absolute Trend	Sibling1	Hl.m. Praha has no clear trend.
Absolute Trend	Sibling2	north Moravia: north Moravia has no clear trend. north Bohemia: north Bohemia has no clear trend. central Bohemia: central Bohemia has no clear trend. south Bohemia: south Bohemia has no clear trend. west Bohemia: west Bohemia has no clear trend. east Bohemia: east Bohemia has no clear trend. south Moravia: south Moravia has no clear trend.
modality	Base	Hl.m. Praha have no clear Modality.
modality	Sibling1	Hl.m. Praha have no clear Modality.
modality	Sibling2	north Moravia has no clear Modality. north Bohemia has no clear Modality. central Bohemia has no clear Modality.

Figure 3.7 Chart Visualizations and models windows

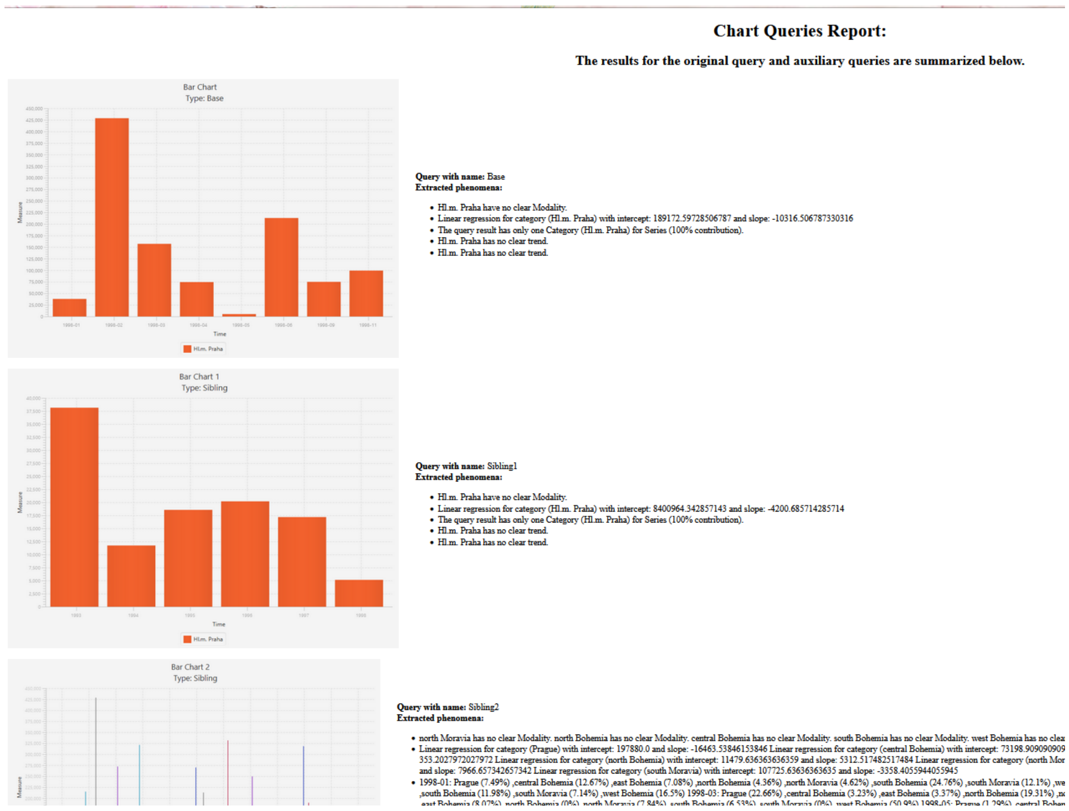


Chart Queries Report:

The results for the original query and auxiliary queries are summarized below.

Query with name: Base
Extracted phenomena:

- Hl.m. Praha have no clear Modality.
- Linear regression for category (Hl.m. Praha) with intercept: 189172.59728506787 and slope: -10316.506787330316
- The query result has only one Category (Hl.m. Praha) for Series (100% contribution).
- Hl.m. Praha has no clear trend.
- Hl.m. Praha has no clear trend.

Query with name: Sibling1
Extracted phenomena:

- Hl.m. Praha have no clear Modality.
- Linear regression for category (Hl.m. Praha) with intercept: 8400964.342857143 and slope: -4200.685714285714
- The query result has only one Category (Hl.m. Praha) for Series (100% contribution).
- Hl.m. Praha has no clear trend.
- Hl.m. Praha has no clear trend.

Query with name: Sibling2
Extracted phenomena:

- north Moravia has no clear Modality. north Bohemia has no clear Modality. central Bohemia has no clear Modality. south Bohemia has no clear Modality. west Bohemia has no clea
- Linear regression for category (Prague) with intercept: 197880.0 and slope: -16463.53846153846 Linear regression for category (central Bohemia) with intercept: 73198.909090909 353.2027972027972 Linear regression for category (north Bohemia) with intercept: 11479.636363636359 and slope: 5312.517482517484 Linear regression for category (north Mor and slope: 7966.657342657342 Linear regression for category (south Moravia) with intercept: 107725.63636363635 and slope: -3358.4055944055945
- 1998-01: Prague (7.49%) ,central Bohemia (12.67%) ,east Bohemia (7.08%) ,north Bohemia (4.36%) ,north Moravia (4.62%) ,south Bohemia (24.76%) ,south Moravia (12.1%) ,we ,south Bohemia (11.98%) ,south Moravia (7.14%) ,west Bohemia (16.5%) 1998-03: Prague (22.66%) ,central Bohemia (3.23%) ,east Bohemia (3.37%) ,north Bohemia (19.31%) ,no ,east Bohemia (8.07%) ,north Bohemia (0%) ,north Moravia (7.84%) ,south Bohemia (6.53%) ,south Moravia (0%) ,west Bohemia (50.9%) 1998-05: Prague (1.29%) ,central Bohen

Figure 3.8 Preview of Report.html

3.2 Extending Delian Cubes for Chart-Query Execution

The implementation of the chart query in Delian is described below, based on each stage of preprocessing and execution, until the extraction of the results.

3.2.1 Necessary programming tools for deployment of DelianCubeEngine

The project was developed within the Eclipse environment, with its core code written in Java 1.8, while uses maven 3.8.1 as a building tool for its deployment. JavaFX graphic packages are utilized for creating graphical interfaces and representations. The system supports a MySQL 8.0.28 database.

3.2.2 Conversion of User Input to Chart Request Object

The class ChartQueryEditorController.java is responsible for handling the user 's input fields and converting them into an expression that can be parsed and executed from the AnalyzeOperator in the server. More specifically, method constructQuery in the file ChartQueryEditorController.java constructs the following expression:

ANALYZE <aggrfunc>(<measure>) FROM <cubeName> FOR <list of atomic filters> GROUP BY <grouper$_1$, grouper$_2$> AS first_query

Below is described how the method works in reference to the figure 3.1.4:

The field aggrfunc, is the "Select Aggregation Function" option chooser with available options {MIN, MAX, AVG, SUM}. The measure field is the text field "Measure Column(y-axis)", the cubeName is the text field "Cube Name". The list of atomic fields is created by determine the first atomic field in the input text fields "Filter Column Name" and "Filter Column Value". Any additional atomic field can be added, by clicking the button "Add Additional Filter" and inserting input in the fields "Additional Column Name" and "Additional Column Value". The fields grouper$_1$, grouper$_2$ are determined by the text fields "Data series grouper 1(x-axis)" and the "Data series grouper 2". Moreover, it is worth noting that the 'analyze' package can compute an expression for more than 2 groupers. However, for chart design purposes in the two-dimensional space, we are restricted to specifying only 2 groupers. Finally, the "Specify chart" radio button is used to save the selected option for chart with available options {Bar chart, Scatter plot, Line Chart} and handler method is the plot Selected.

The number of input parameters in the Chart Query Editor window required to execute a chart query is presented in the table below:

Input Parameter Name	Corresponding Analyze Expression Parameter	Number Of Input Parameter	Range Value of Input Parameter
'Select Aggregation Function'	aggrfunc	1	{AVG, MIN, MAX, SUM}
'Measure Column (y-axis)'	measure	1	Any valid word (numeric column in DB)
'Cube Name'	cubeName	1	Any valid word (cube schema in DB)
'Filter Column Name' 'Filter Column Value'	list of atomic filters	1...*	Any valid pair field-value(column and value contained in column in DB)
'Data Series Grouper 1 (x-axis)' Data Series Grouper 2	list of groupers (grouper$_1$, grouper$_2$)	2	grouper$_1$ (Date column) grouper$_2$ (Any column)
'Specify chart'	-	1	{Bar chart, Scatter plot, Line Chart }

Table 3.2.1 Input Parameters of ChartQueryEditor

For the creation of the chart and the computation of extracted models we need to transfer the analyze expression and the chart selected option from client to server in a more compact way. For this purpose, a new object "ChartRequest" has been created.

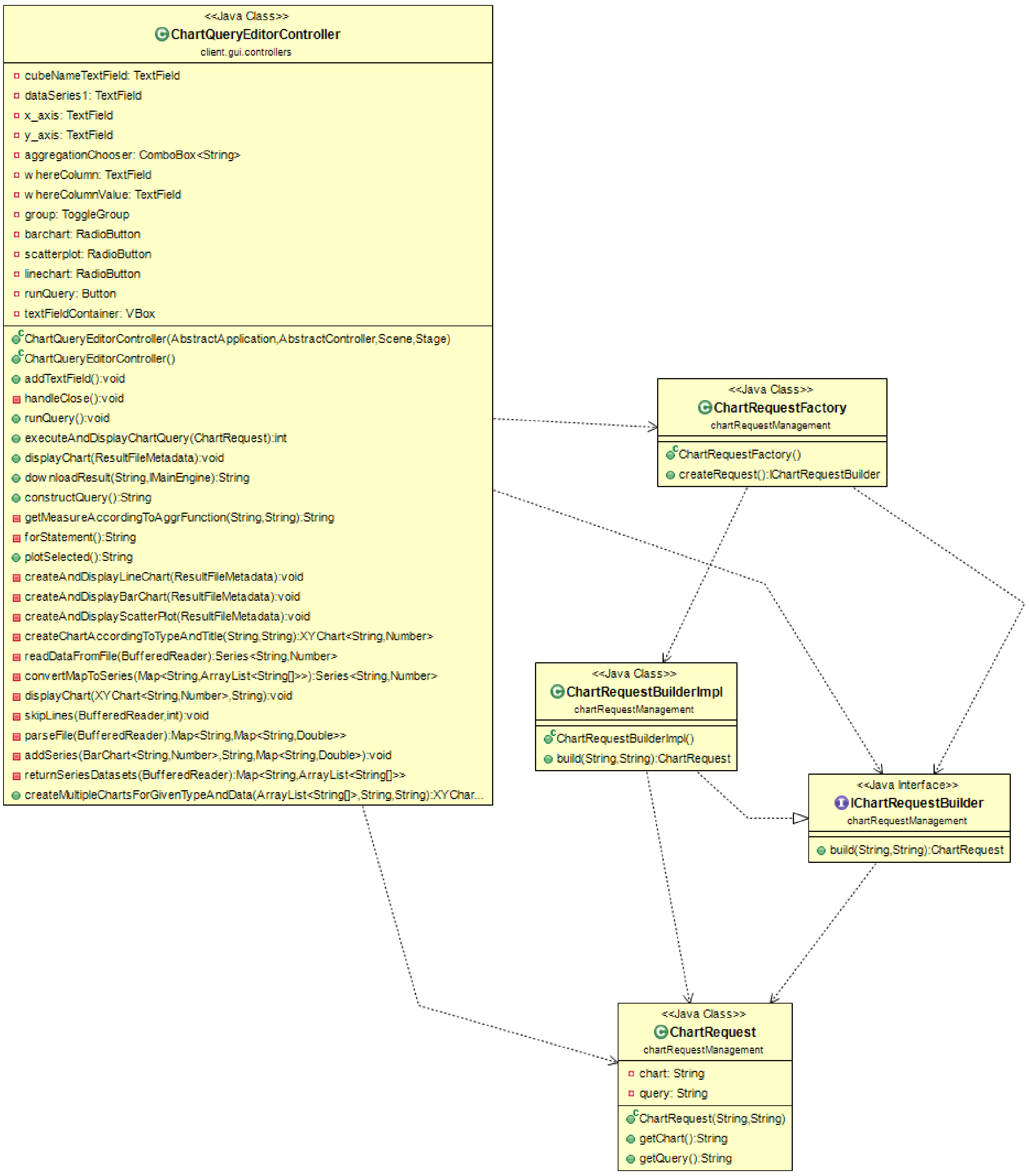
3.2.3 Architecture of Chart Request Object (chartRequestManagement package)

As mentioned in the previous paragraph, the chart request object was created to transfer information from the client package to the server package in a more compact manner. In the future, the ability to create more complex chart requests could be added, containing additional parameters beyond the currently provided chart selection and query (e.g., allowing users to select specific statistical models they want to see). For this purpose, the architecture of chartRequestManagement package uses the Builder design pattern. Secondarily, the Factory design pattern is used, to make easier the addition of "new" chartRequestBuilders.

The chartRequestManagement package contains the classes ChartRequest, ChartRequestFactory, ChartRequestBuilderImpl and the interface IChartRequestBuilder.

The ChartRequest class is the object/entity that contains the user' query and selected chart option. Furthermore, it implements the Serializable Interface as the Delian project uses RMI (Remote Method Invocation) server, and because of that it can send and accept only serializable objects. The ChartRequestFactory creates IChartRequestBuilder objects, practically is responsible for creating the different Builders that will create differently the chart requests. The ChartRequestBuilderImpl is the concrete and only one implementation of the IChartRequestBuilder for the time being. Finally, the interface IChartRequestBuilder functions as a contract that all concrete Builders must implement.

The overall architecture of chartRequestManagement package with the addition of client's class ChartQueryEditorController in the form of UML diagram is demonstrated in Figure 3.9.



<<Java Class>>
ChartQueryEditorController
client.gui.controllers
cubeNameTextField: TextField
dataSeries1: TextField
x_axis: TextField
y_axis: TextField
aggregationChooser: ComboBox<String>
whereColumn: TextField
whereColumnValue: TextField
group: ToggleGroup
barchart: RadioButton
scatterplot: RadioButton
linechart: RadioButton
runQuery: Button
textFieldContainer: VBox
ChartQueryEditorController(AbstractApplication,AbstractController,Scene,Stage)
ChartQueryEditorController()
addTextField():void
handleClose():void
runQuery():void
executeAndDisplayChartQuery(ChartRequest):int
displayChart(ResultFileMetadata):void
downloadResult(String,IMainEngine):String
constructQuery():String
getMeasureAccordingToAggrFunction(String,String):String
forStatement():String
plotSelected():String
createAndDisplayLineChart(ResultFileMetadata):void
createAndDisplayBarChart(ResultFileMetadata):void
createAndDisplayScatterPlot(ResultFileMetadata):void
createChartAccordingToTypeAndTitle(String,String):XYChart<String,Number>
readDataFromFile(BufferedReader):Series<String,Number>
convertMapToSeries(Map<String,ArrayList<String[]>>):Series<String,Number>
displayChart(XYChart<String,Number>,String):void
skipLines(BufferedReader,int):void
parseFile(BufferedReader):Map<String,Map<String,Double>>
addSeries(BarChart<String,Number>,String,Map<String,Double>):void
returnSeriesDatasets(BufferedReader):Map<String,ArrayList<String[]>>
createMultipleChartsForGivenTypeAndData(ArrayList<String[]>,String,String):XYChar...
<<Java Class>>
ChartRequestFactory
chartRequestManagement
ChartRequestFactory()
createRequest():IChartRequestBuilder
<<Java Class>>
ChartRequestBuilderImpl
chartRequestManagement
ChartRequestBuilderImpl()
build(String,String):ChartRequest
<<Java Interface>>
IChartRequestBuilder
chartRequestManagement
build(String,String):ChartRequest
<<Java Class>>
ChartRequest
chartRequestManagement
chart: String
query: String
ChartRequest(String,String)
getChart():String
getQuery():String

Figure 3.9 Overall architecture of chartRequestManagement

3.2.4 Method answerCubeQueryFromChartRequest in IMainEngine and SessionQueryProcessorEngine.

From ChartQueryEditorController (client class) the ChartRequest object is transferred via the call of method answerCubeQueryFromChartRequest. Like all existing methods that make a call from the client to server in Delian, the method answerCubeQueryFromChartRequest must also be declared in the interface IMainEngine, while its implementation is found in the SessionQueryProcessorEngine, a class that implements the interface IMainEngine. The method accepts a ChartRequest parameter and returns a ResultFileMetadata object (Delian ‘s exististing class that implements Serializable interface).
Practically, the method answerCubeQueryFromChartRequest sequentially performs four tasks. At first, the method initializes a connection between SessionQueryProcessor and ChartManager. Secondly, initializes the connection between between ChartManager and AnalyzeOperator. The connection with AnalyzeOperator is required, because the AnalyzeOperator will process the analyze expression (query) and execute to provide results. After that, the method initializes the ChartManager ‘s fields with the necessary information (type and query) provided by ChartRequest parameter. Finally, method generates and executes the chartQueries via the calls of ChartManager ‘s methods generateQueries and executeQueries.

3.2.5 Reporting of ChartQueries in File Chart_Queries_Report.md

With the completion of field initialization, all the information necessary to generate and execute queries by the Analyzer is available. Subsequently, these queries can be reported in the Chart_Queries_Report.md file. Initially ChartManager’s method generateQueries(), calls the method execute() of AnalyzeOperator that will parse the expression-query and produce analyze-queries of 3 types basic, sibling and drill-down. In the master thesis, we are only interested in the types basic and sibling. So, we remove from ChartManager ‘s producedQueries list the drill-down queries. The reason for that, is that we want to be able to extract conclusions for phenomena in

the same level of detail with the original basic. The drill-down queries provide a view to a more increased level of detail but not in the same level with the original. After that, ChartManager calls the method reportChartQueriesDetails() of the VisualizationManager class. Method reportChartQueriesDetails() is the main Method of VisualizationManager, taking as parameters a list of ChartQueries (produced By Analyzer operator) and the user's selected type chart, and is responsible for reporting in file Chart_Queries_Report.md. The method starts with initializing a FileWriter object (java.io.FileWriter) in order to print the results in a file. To initialize the FileWritter, we must determine a filename and a filepath. For the time being, filename and filepath are fixed with values: File-ChartQueries_Report.md and OutputFiles.
For every query result produced we print in File-ChartQueries_Report.md the following info:

- **Visualization:** default or small multiplies.
- **Type:** Base or Sibling.
- **Details:** -/ what changed in comparison with the base query.
- **An array with 3 columns: Grouper 1, Grouper 2, Measure** containing the results.
- **X_axis values:** the distinct values for x_axis (date values of grouper 1).
- **Series:** the distinct categories from Grouper 2.

Next, we break down the utility of every piece of info in file:
Visualization with value default, means that for the different categories contained in series, one chart will be created. Visualization with value small multiplies means that will be created one figure with many small charts, one for each distinct category containing in series. Method decideVisualizationType() of VisualizationManager is responsible for determining the value(default or small multiplies). The method returns "small multiplies" if the number of distinct categories in array is greater than 5, else "default". The number 5 may seem arbitrary and is indeed, but it was decided for the quicker and effective processing of results from human eye.
Type with value basic or sibling as these are the two types of analyze queries that we process. The type is required for the title of every chart.

Details with value an empty string for basic query or details for what changed in filter value and grouper for the produced sibling query. Details are required for determining for every additional chart query produced (sibling) what has changed in conditions (filters and groupers), in comparison to basic query.
An array with 3 columns: Grouper 1, Grouper 2, Measure with dimension (**number of results produced**) **X 3.** The results produced for every query. Containing a point represented in x-axis from value in Grouper1 column and in y-axis from Measure column. The Grouper 2 is used for the group of values in a category.
X_axis values with value the distinct date values for Grouper 1. For laying the x-axis values in graph.
Series: the distinct categories from Grouper 2. For keeping the info of how many distinct categories has the produced chart.

3.2.6 Method answerCubeQueryFromChartRequestAndReturnAsChartResponse in IMainEngine and SessionQueryProcessorEngine.

From ChartQueryEditorController (client class) the ChartRequest object is transferred via the call of method answerCubeQueryFromChartRequestAndReturnAsChartResponse in the case that **the result is a unique time-series**. Like the method answerCubeQueryFromChartRequest must also be declared in the interface IMainEngine, while its implementation is found in the SessionQueryProcessorEngine, a class that implements the interface IMainEngine. The method accepts a ChartRequest parameter and returns a ChartResponse object (Delian ‘s new class that implements Serializable interface and is saved under path chartManagement/utils).
Practically, the method answerCubeQueryFromChartRequestAndReturnAsChartResponse sequentially performs the same four tasks with answerCubeQueryFromChartRequest. Their difference is that answerCubeQueryFromChartRequest reports the necessary info for chart visualizations, models e.t.c into an .md file and returns the ResultFileMetadata object, which contains the location and the name of the file, to the client but answerCubeQueryFromChartRequestAndReturnAsChartResponse returns immediately to the client a new serializable Object ChartResponse containing the necessary info chart visualizations, models and **scores of highlights**.
Furthermore, **ChartResponse class contains 2 fields:**

- **List of ChartVisModel** items
- **List of ChartScoreModel** items.

ChartVisModel class is a helper class, containing all the necessary info for the chart visualization of the query in the client. More specifically:

- List of DataPoint items: helper class to hold the information of the triple <grouper1, grouper2, measure> for every point of chart.
- ChartVisType: to hold the chart type that was selected in ChartQueryEditor window
- QueryType: to hold the type of query Base or Sibling$_i$ (where i is index number in the list of siblings)
- List of x_axis_values: to hold the x_axis values
- SQLexpression: to hold the cube query's corresponding sql expression

ChartScoreModel class is a helper class, containing all the necessary info for the score of extraction algorithm model in the client. More specifically:

- Score: a double value showing how important a highlight is.
- Name: the name of the model
- ChartVisModel: The chartVisModel object that belongs to.
- Result: A string result that contains the result of the extraction algorithm model.

3.2.7 Computation And Reporting of Models for ChartQueries

After reporting of chartQueries in the file File-ChartQueries_Report.md (for multiple time-series) or creating the list of ChartVisModel via the call reportChartQueryDetailsForChartResponse in VisualizationManager (for unique-timeseries), the ChartManager calls the suitable method of ModelManager class.
For (multiple time-series) ChartManager calls the method reportModelsForChartType. The method starts with initializing a FileWriter object to print the results extracted from models in the file File-ChartQueries_Report.md. The result of every model is computed with the call of compute method, that returns 0 if it has successfully parsed the results array from file File-ChartQueries_Report.md. After that, we print the results as 2D string array that has 3 columns for each model executed for every query with column names: Model, Type and Result.

Briefly, every column in the array contains the following info:

- **Model:** containing the name of the model that was executed.
- **Type:** type of query (basic or sibling)
- **Result:** the result of the model for the query' results.

For (unique time-series) ChartManager calls the method getScoreModelsFor-ChartVisModels. The method takes as parameter the List of ChartVisModel items and returns a list of ChartScoreModel items. For every new ChartScoreModel that is created the 4 fields: **score, name, chartVisModel and result** are initialized accordingly. The score is computed via the call of method computeScore of ChartModel and returns a double value in the range [-1,1] as it was described for every model in 3.1.4 Queries and Models.

Next, we describe how every model sets the result for the query:

ContributorModel: Initially, because the results for basic and sibling queries have been saved in the same 2-dimensional array, we create a list containing smaller two-dimensional arrays, one for every query. The small array contains the query's results, so we pass it from method findContributionInArray().
If the query's results have only one distinct value for grouper 2 (one unique series) then the result that is returned is: "Series has a mega contributor for x ='value of the grouper_1 with greatest measure'. Else, for multiple series, we return the series with the sum(max_measure) for every grouper_1 value.

AbsoluteTrendModel: Initially, as previously, we create a list containing smaller two-dimensional arrays, one for every query. The small array contains the query's results, so we pass it from method findTrendInArray().
If the query's results contain measures such that from older to newer date the measure is increasing, then we have an uptrend. The returned result is "Series has an absolute uptrend.". In the opposite case, that from older to newer date the measure is decreasing then we have a downtrend. The returned result is "Series has an absolute downtrend.". If nothing from above facts is valid then the result that is returned is "Series has not a clear trend.".

KendallBasedTrendModel: Likewise AbsoluteTrend model, but computes the correlation between x_values and measures, using the Kendall's coefficient. Then if Kendall's coefficient ≥ 0.5 and Kendall's coefficient< 1 returns "Series name" has an uptrend. Else if Kendall's coefficient ≥ -0.5 and Kendall's coefficient< 0.5 1 returns "Series name" has no clear trend. Else "Series name" has a downtrend

ModalityModel: Initially, as previously, we create a list containing smaller two-dimensional arrays, one for every query. The small array contains the query's results, so we pass it from method findModalityInArray().
For every query list of measures, we create a new list that contains the differences between sequential x-axis values. After that, we iterate through the list, and we increment a counter if the product of two sequential differences is negative. The reason for that action is that if two sequential differences have product negative, the monotony of graph has changed (we had ascending order and instantly after descending order or the opposite). In the end: if the number of changes equals 1 then we Unimodality meaning that we had only one extreme point in graph, so we return "has Unimodality". Else if the number of changes equals 3, then we have Bimodality, because we have detected two valleys or two peaks. In any other case we return "has no clear modality".

RegressionModel: Initially, as previously, we create a list containing smaller two-dimensional arrays, one for every query. The small array contains the query's results, so we pass it from method findRegressionInArray().
For every query list of measures and for every series we perform linear regression provided from Apache library [Apac99]. Apache Library provide the SimpleRegression class and via the method add() we can add every point (x,y) of the series.To get the linear's regression intercept we use method getIntercept() and for slope we use method getSlope(). In the end, for every category it is returned as a result: "Linear regression for series (series name) with intercept: regression's intercept and slope: regression's slope."

3.2.8 Architecture of ChartManagement package

The chartManagement package is designed to facilitate the creation of chart visualization in the client package and the computation of extracted phenomena from the data. Internally contains the packages models and utils (demonstrated in Figure 3.10). Package utils has been created with purpose to contain helper classes, and for the time being it only contains the class DataPoint, that is used for the representation of a data point in the results array. The models package contains the forementioned model classes and the classes ChartModel, ModelListFactory and ModelManager.

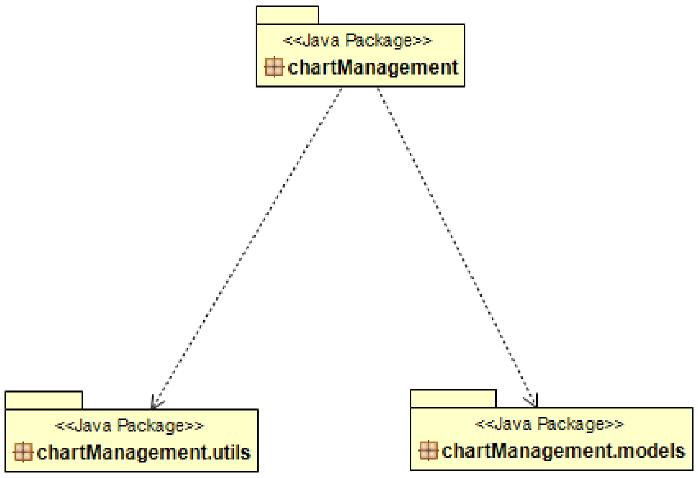


Figure 3.10 Subpackages of chartManagement

The architecture of the package models utilizes the Factory Method design pattern to manage the creation of ChartModel objects. The abstract class ChartModel serves as the base class, defining the common interface and behavior for all ChartModel types. The concrete subclasses: ContributorModel, DominanceModel, ModalityModel, RegressionModel and TrendModel, extend this abstract class, each providing specific implementations of the required abstract methods with most important the abstract compute() method. The ModelListFactory class encapsulates the instantiation logic within its createModelsForChartType() method, which returns a list of ChartModel objects. This method instantiates various subclasses of ChartModel and aggregates them into a single collection. The single collection that is returned depends on the method 's parameter (IChartQueryNModelGenerator object). The method call is made by the ModelManager class, which acts as a coordinator for the package models. The architecture of models package is demonstrated in Figure 3.11.

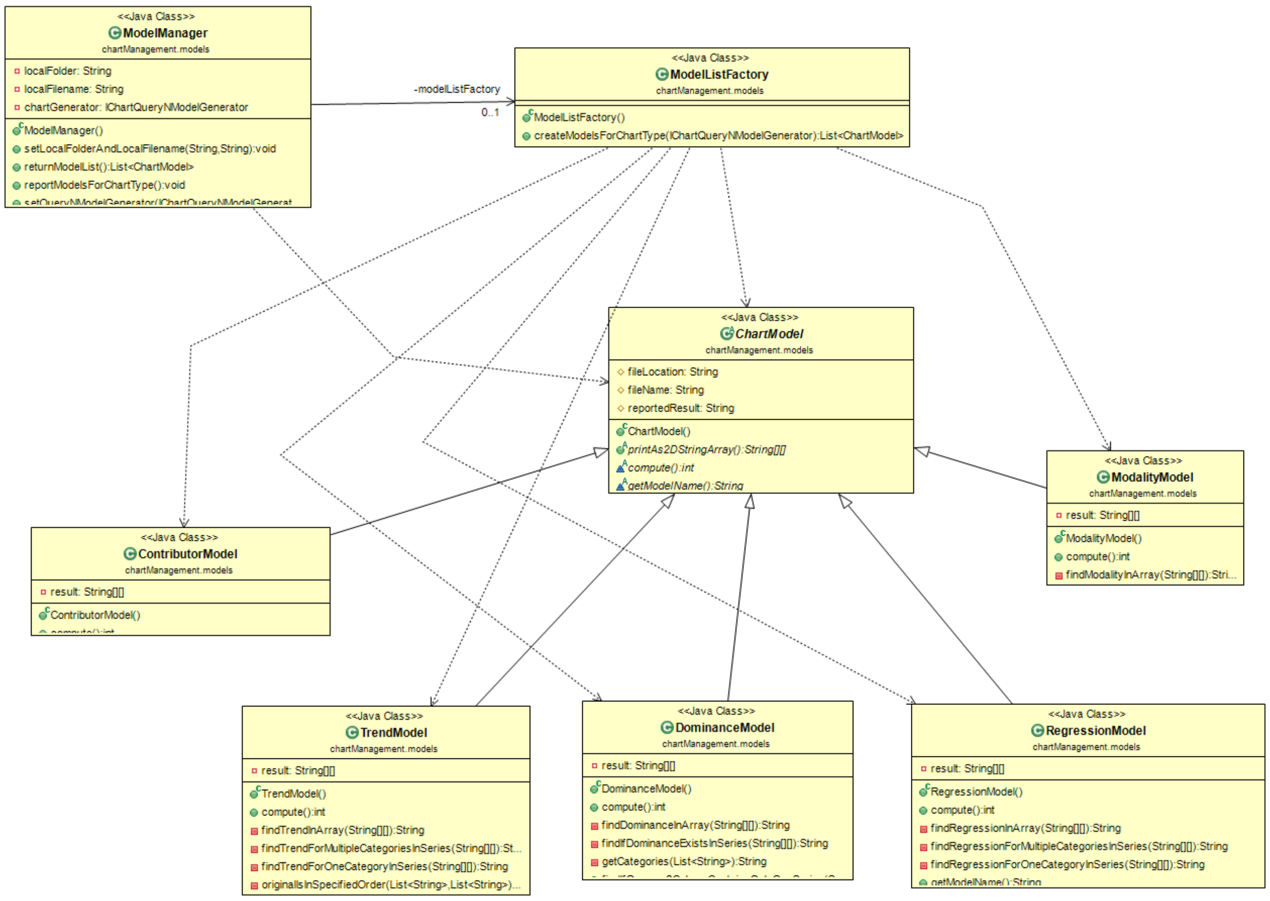
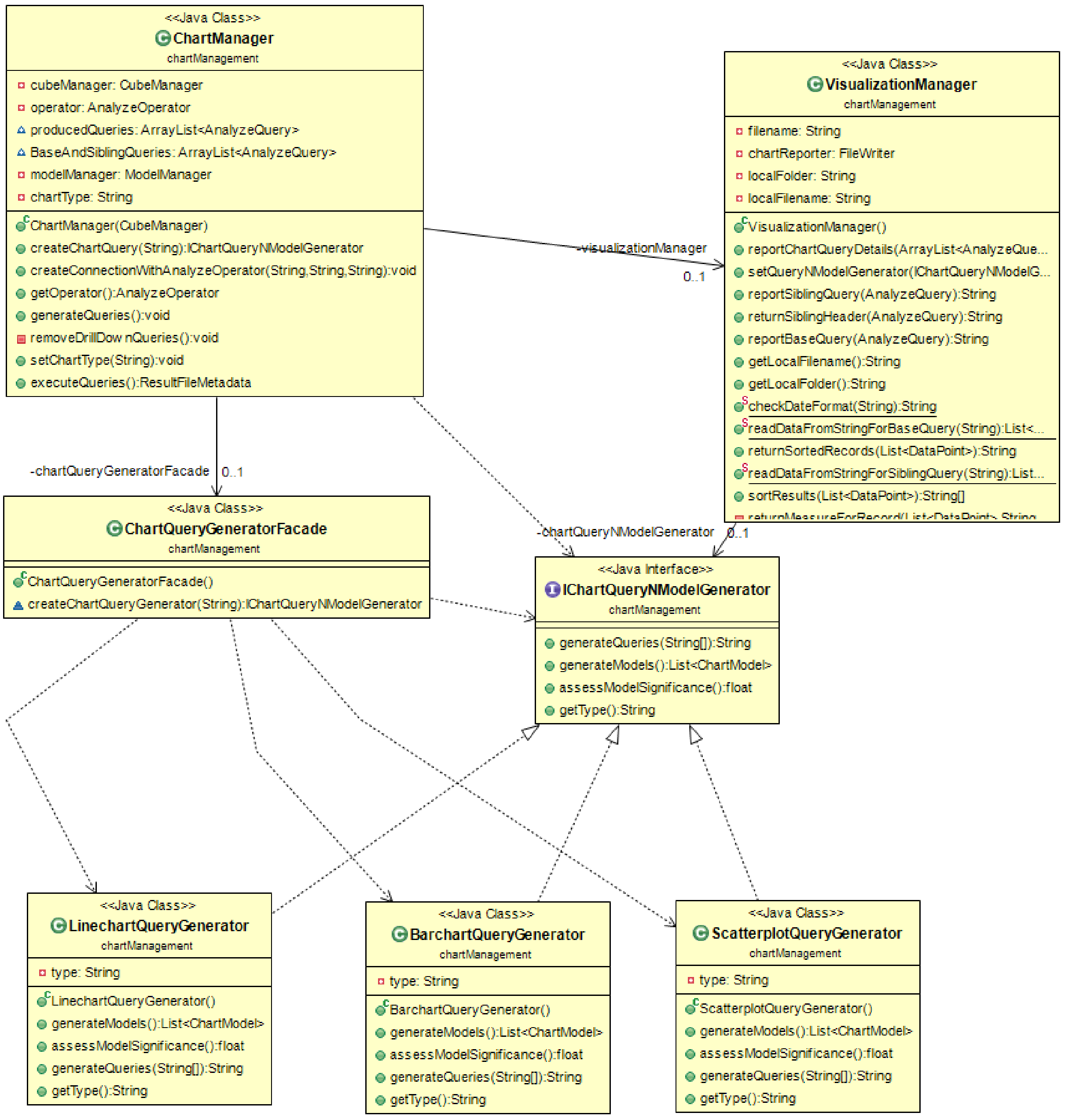


Figure 3.11 Architecture of package models.

Apart from the subpackages models and utils, the ChartManagement package contains the classes: ChartManager, VisualizationManager, ChartQueryGeneratorFacade, LinechartQueryGenerator, ScatterplotQueryGenerator and BarchartQueryGenerator and the interface IChartQueryNModelGenerator.

The connection between classes leverages both the Factory Method and Strategy design patterns. The IChartQueryNModelGenerator interface defines the contract for the different types of charts, implemented by concrete classes LinechartQueryGenerator, ScatterplotQueryGenerator and BarchartQueryGenerator. The ChartQueryGeneratorFacade class encapsulates the creation logic, producing instances of the IChartQueryNModelGenerator interface based on specified string type of chart. The ChartManager class holds a reference to an IChartQueryNModelGenerator object and delegates the execution of the methods to the IChartQueryNModelGenerator and furthermore uses the ChartQueryGeneratorFacade to create the object in the first place. Within the VisualizationManager class, the reportChartQueryDetails method is executed based on the selection of the appropriate IChartQueryNModelGenerator object, provided by the ChartManager (Figure 3.12).

Figure 3.11 Test for constructQuery in ChartQueryEditorController class.



<<Java Class>>
ChartManager
chartManagement
cubeManager: CubeManager
operator: AnalyzeOperator
producedQueries: ArrayList<AnalyzeQuery>
BaseAndSiblingQueries: ArrayList<AnalyzeQuery>
modelManager: ModelManager
chartType: String
ChartManager(CubeManager)
createChartQuery(String):IChartQueryNModelGenerator
createConnectionWithAnalyzeOperator(String,String,String):void
getOperator():AnalyzeOperator
generateQueries():void
removeDrillDownQueries():void
setChartType(String):void
executeQueries():ResultFileMetadata
-visualizationManager
0..1
<<Java Class>>
VisualizationManager
chartManagement
filename: String
chartReporter: FileWriter
localFolder: String
localFilename: String
VisualizationManager()
reportChartQueryDetails(ArrayList<AnalyzeQue...
setQueryNModelGenerator(IChartQueryNModelG...
reportSiblingQuery(AnalyzeQuery):String
returnSiblingHeader(AnalyzeQuery):String
reportBaseQuery(AnalyzeQuery):String
getLocalFilename():String
getLocalFolder():String
checkDateFormat(String):String
readDataFromStringForBaseQuery(String):List<...
returnSortedRecords(List<DataPoint>):String
readDataFromStringForSiblingQuery(String):List...
sortResults(List<DataPoint>):String[]
-chartQueryGeneratorFacade
0..1
<<Java Class>>
ChartQueryGeneratorFacade
chartManagement
ChartQueryGeneratorFacade()
createChartQueryGenerator(String):IChartQueryNModelGenerator
-chartQueryNModelGenerator
0..1
<<Java Interface>>
IChartQueryNModelGenerator
chartManagement
generateQueries(String[]):String
generateModels():List<ChartModel>
assessModelSignificance():float
getType():String
<<Java Class>>
LinechartQueryGenerator
chartManagement
type: String
LinechartQueryGenerator()
generateModels():List<ChartModel>
assessModelSignificance():float
generateQueries(String[]):String
getType():String
<<Java Class>>
BarchartQueryGenerator
chartManagement
type: String
BarchartQueryGenerator()
generateModels():List<ChartModel>
assessModelSignificance():float
generateQueries(String[]):String
getType():String
<<Java Class>>
ScatterplotQueryGenerator
chartManagement
type: String
ScatterplotQueryGenerator()
generateModels():List<ChartModel>
assessModelSignificance():float
generateQueries(String[]):String
getType():String

Figure 3.12 Connection between main classes of ChartManagement package.

3.3 Validation of the System

The correctness of the main methods used in the package ChartManagement and client's package new file: ChartQueryEditor.java is verified through unit tests. The unit tests are executed using JUnit 4, and Mockito and for the file ChartQueryEditor.java that uses extensively the JavaFX library, TestFX is used for the testing of the main method.
All the test cases use the structure *Arrange-Act-Assert.* In the Arrange phase, we set up the preconditions and initialize the objects or resources that the test requires. In the Act phase, we perform the actual action or behavior that we want to test, practically the execution of the method we want to test. In the last phase, the Assert we verify that the action taken in the "Act" phase produces the expected results.

Test for method constructQuery in the class ChartQueryEditor

The method constructQuery is responsible for gathering the user's input and converting it into an expression that is executable from the analyze operator. We tested if we insert info in the different fields of the ChartQueryEditor window and after call the method constructQuery if the returned constructed query is the same with the expected query. The result of the test is demonstrated in the Figure 3.3.1.

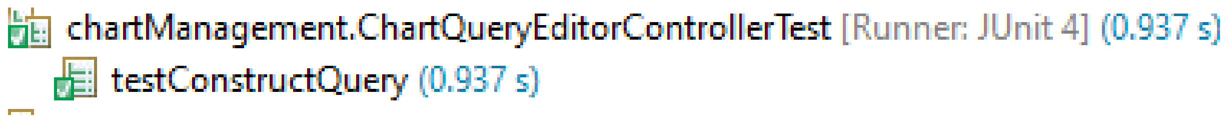

chartManagement.ChartQueryEditorControllerTest [Runner: JUnit 4] (0.937 s)
testConstructQuery (0.937 s)

Figure 3.11 Test for constructQuery in ChartQueryEditorController class.

Test for methods reportSiblingQuery and reportBaseQuery in the VisualizationManager.

The methods reportSiblingQuery and reportBasicQuery are responsible for reporting the results for a query of type Sibling and Basic, respectively. In tests testReportBaseQuery and testReportSiblingQuery, we test the scenario that given input in a specific format and some grouper and sigma values as input, the reported String that is returned is the expected. The results of the tests are demonstrated in the Figure 3.11.

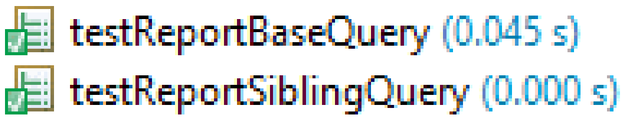

testReportBaseQuery (0.045 s)
testReportSiblingQuery (0.000 s)

Figure 3.12 Tests for reportBaseQuery and reportSiblingQuery in VisualizationManager.

Tests for method checkDateFormat in the VisualizationManager.

The method checkDateFormat in VisualizationManager is responsible for returning the format of the Date passed. Two formats are supported so far "yyyy-MM", "yyyy". For the testing of the method, we pass as input to the method real Dates in the specific format and check if the right form is returned from method. The results of the tests are demonstrated in the Figure 3.12.

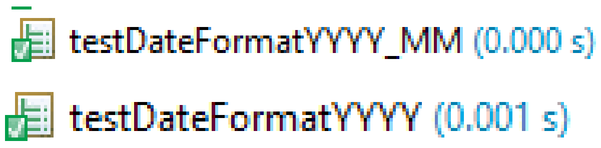

testDateFormatYYYY_MM (0.000 s)
testDateFormatYYYY (0.001 s)

Figure 3.13 Tests for dateFormat in VisualizationManager.

Test for method returnSiblingHeader in the VisualizationManager.

The method returnSiblingHeader in VisualizationManager is responsible for returning the header of a sibling query. The sibling header contains details about the changes in the groupers and filters from the original "basic" query. In the test named testReturnSiblingHeader we test if these changes are passed correctly. The result of the test is demonstrated in the Figure 3.13

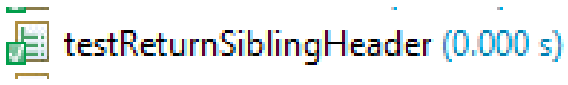
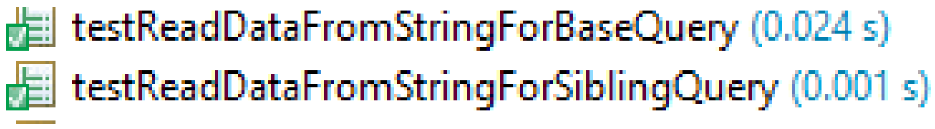

testReturnSiblingHeader (0.000 s)

Figure 3.24 Test for returnSiblingHeader in VisualizationManager.

Test for methods readDataFromStringForBaseQuery and readDataFromStringForSiblingQuery in the VisualizationManager

The methods readDataFromStringForBaseQuery and readDataFromStringForSiblingQuery are responsible for reading the string results for Base and Sibling queries, respectively and returning an array of DataPoints for their visualization. However, the string results contain and other details and not only DataPoints. The tests testReadDataFromStringForSiblingQuery, testReadDataFromStringForBaseQuery check that the result array is the correct one and for both methods we skip the correct number of lines. The result of the tests is demonstrated in the Figure 3.14

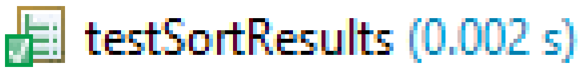


```
testReadDataFromStringForBaseQuery (0.024 s)
testReadDataFromStringForSiblingQuery (0.001 s)
```

Figure 3.15 Tests for readDataFromStringForBaseQuery and readDataFromStringForSiblingQuery in VisualizationManager.

Test for method sortResults in the VisualizationManager

The method sortResults is responsible for sorting the input DataPoints in ascending order according to grouper1 (x-axis). In test we give as input an array with arbitrary order and we expect the method sortResults to sort the array in ascending order. The result of the test is demonstrated in Figure 3.15.



```
testSortResults (0.002 s)
```

Figure 3.16Test for sortResults in VisualizationManager.

Test for method decideType in the VisualizationManager

The method decideType is used to determine whether the visualization in a Barchart will be "small multiplies" or "default". In the implementation code we have determine that if the series number is greater than 5, small multiplies is selected otherwise default. The result of the test is demonstrated in Figure 3.16.

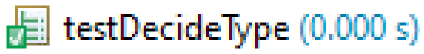

testDecideType (0.000 s)

Figure 3.17 Test for decideType in VisualizationManager.

Test for processResultsForVisualization in VisualizationManager

Similarly, with the decideType, but it takes more parameters and calls internally the decideType. The result of the test is demonstrated in Figure 3.17.

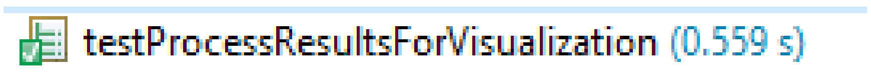

testProcessResultsForVisualization (0.559 s)

Figure 3.18 Test for processResultsForVisualization in VisualizationManager.

Test for returnModelList in the ModelManager

The method returnModelList is implemented in ModelManager class and returns a list with models according to the field chartGenerator of type IChartQueryNModelGenerator. In the test, we check if we set the field's type: BarchartQueryGenerator (concrete implementation of IChartQueryNModelGenerator) if the list will return the models Dominance and Contributor, which are determined from the ModelListFactory. The result of the test is demonstrated in 3.18.

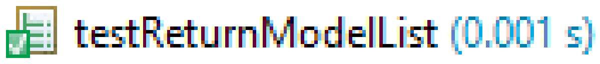

testReturnModelList (0.001 s)

Figure 3.19 Test for returnModelList in ModelManager.

Test for findContributionInArray in ContributorModel

The method findContributionInArray is implemented in ContributorModel class and returns a String result that contains the percentage (%), for every category in series that contributes to the result. In the case, that a series contain only one category, then we have 100% contribution to the result. In the test, we check for two different query results, one with one category and one with two categories and a dominator category, if method findContributionInArray will return the right percentage. The result of the test is demonstrated in 3.19.

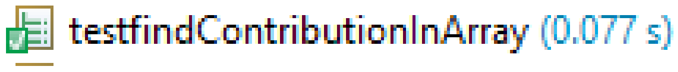

testfindContributionInArray (0.077 s)

Figure 3.20 Test for findContributionInArray in ContributorModel class.

Test for findModalityInArray in ModalityModel

The method findModalityInArray is implemented in ModalityModel class and returns a String result that informs if there is a Unimodality or Bimodality or no Clear Modality in query's results. In the test, we check for these three cases, if method will return the right String result. The result of the test is demonstrated in 3.20.

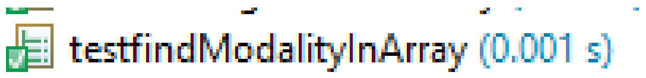

testfindModalityInArray (0.001 s)

Figure 3.21 Test for findModalityInArray in ModalityModel class.

Test for findRegressionInArray in RegressionModel

The method findRegressionInArray is implemented in RegressionModel class and returns a String result that contains the linear regression coefficients: intercept and slope. In the test, we insert specific points (dates,measure) in query's result and we check if linear 's regression coefficients returned from method are the expected ones. The result of the test is demonstrated in 3.21.

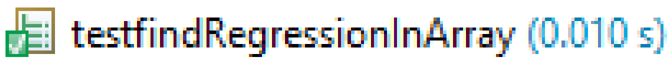

testfindRegressionInArray (0.010 s)

Figure 3.22 Test for findRegressionInArray in RegressionModel class.

Test for findTrendInArray in TrendModel. The method findTrendInArray is implemented in TrendModel class and returns a String result that informs if there is an uptrend or downtrend or no clear trend in query's results. In the test, we check for these three cases, if method will return the right String result. The result of the test is demonstrated in 3.22.

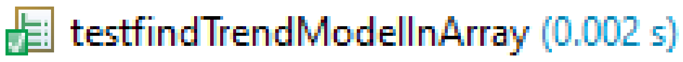

testfindTrendModelInArray (0.002 s)

Figure 3.23 Test findTrendInArray in TrendModel class.

CHAPTER 4

Experiments

In this Chapter, we experimentally evaluate the behavior of the constructed software in relation to the parameters of the data that can affect this behavior.

4.1 Experimental Setup

The experimental objective concerns the study of the execution time of the *Query-As-A-Chart* operator for different queries and data.

For the experimental evaluation of this thesis, three experiments were conducted for three different parameters of the problem. The parameters evaluated are the size of the data set, the number of selection levels of the query. The experiments measured the execution time of the operator from start to finish, as well as the individual execution steps, in order to determine which execution step causes the most delay. The individual execution steps that were timed for result **unique time series** are depicted in Figure 4.1.

Execution Step	Methods	Delian Class
Conversion of User Input to ChartRequest Object	convertUserInput2Char-tRequest	ChartQueryEditorController
Execution of Queries and Report Chart	reportChartQue-ryDetailsForChar-tResponse	VisualizationManager
Execution of Models	getScoreModelsFor-ChartVisModels	ModelManager
Creation of Datastory	createDatastory	ChartQueryEditorController

Table 4.1 Individual Execution Steps for Unique Series.

The execution times presented are the average of five measurements. The system on which the experiments were conducted is Windows 10, with an AMD Ryzen 7 4800H CPU @ 2.90 GHz, 8GB RAM, and a 490GB SSD.

The datasets that were used in the experimental process are the following:

Dataset	Number of Dimen-sions	Number of levels	Number of Tu-ples
Loan Cube (Data-base pkdd99_star_100K)	3	12	100.000
Loan Cube (Data-base pkdd99_star_1M)	3	12	1.000.000
Loan Cube (Data-base pkdd99_star_10M)	3	12	10.000.000

4.2 Impact of the number of filter values on execution time

We conducted three queries for this experiment, which have fixed structure in terms of aggregate function, grouping levels (always 2) and type of chart but have one, two and three levels of selection respectively. The goal is to understand the effect that the number of selection levels may have on the execution time of the operator, keeping the structure of the rest of the query constant, which includes two levels of grouping, type of query, a cube and an aggregate function.

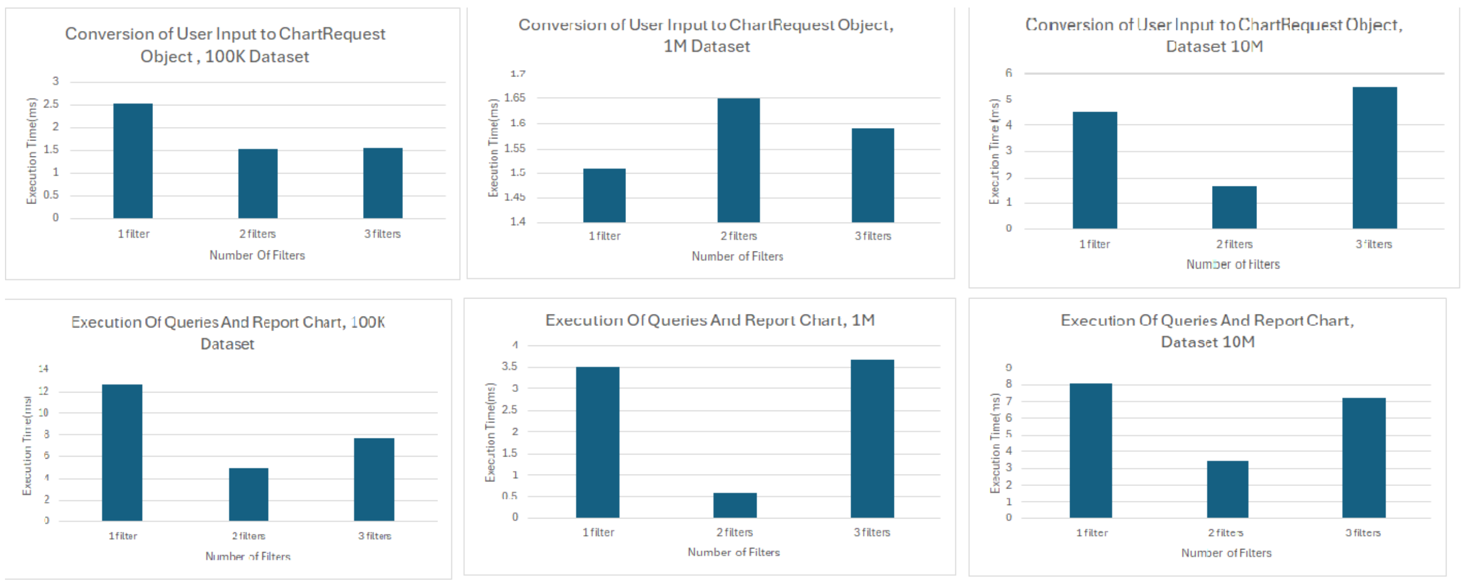


Figure 4.1 Execution Time in ms for the first two individual steps: 1) Conversion of User Input to ChartRequest Object Execution 2) Execution Of Queries And Report Chart.

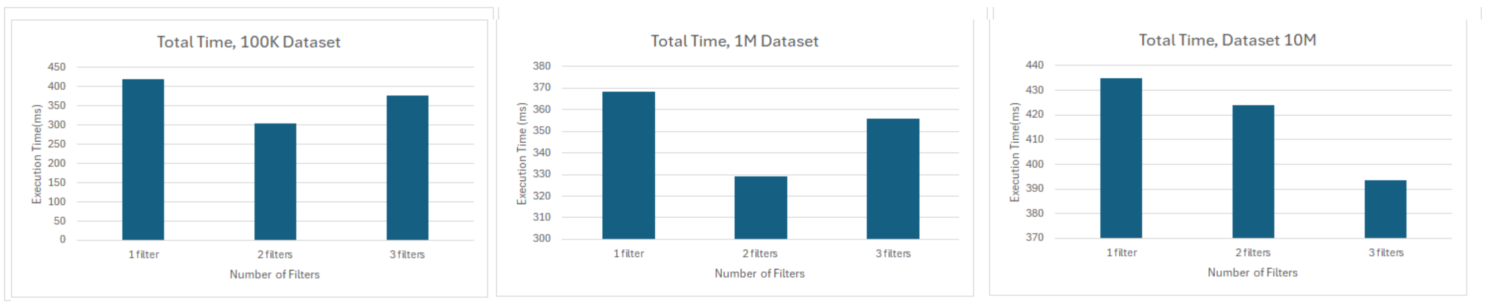


Figure 4.3 Total execution time for each filter and Dataset.

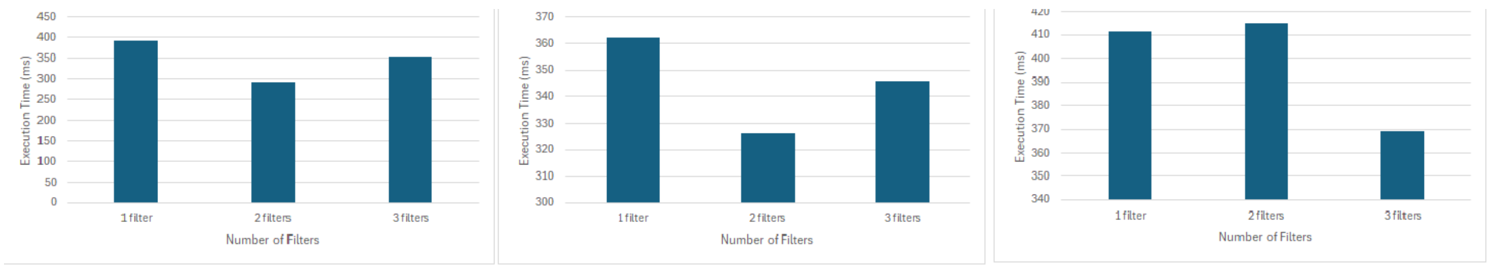


Figure 4.2 Execution Time in ms for the last two individual steps: 3) Execution of Models 4) Creation of Datastory (* the real time measurement is 95% shorter, by accident the save of chart images has been calculated in).

Concerning the first two steps of the processing, it appears that the execution time is pretty much around the same values as the number of filters varies (Figure 4.1) However, for dataset with size 100K and 1M seems that the middle value for filters (filters =2) succeeds the best execution time.
Observing the individual execution steps, we can comment that the conversion of user input to ChartRequest object doesn’t seem to be affected from some logic factors as the number of filters or the size of dataset. As for the execution step ‘Execution of Queries and Report Chart’, seems that for every size of Dataset, the median value for filters (filters=2) leads to smallest execution time. The fact that we always have two groupers, and when we test for two filters, we determine filters that belong to the same dimensions as groupers, leads to shortest execution time from package analyzer.
With a close look at Figure 4.2, we can observe the same phenomenon we notice and for the next execution step ‘Execution of Models’ which seems to be a logic consequence of the previous execution step. Finally, observe that the creation of Datastory for 10M takes a longer time in comparison with the time that takes for 100K and 1M, adding time to the total. It seems reasonable in a way, because it writes more data in the html file and if the selection of the filter was successful, succeeds to extract many matching results from the database.

4.3 Impact of number of Siblings in the Execution Time

For this experiment, we tried to set up queries with many sibling queries and observe if the increase in number of siblings leads to important increase in execution of the time. We conducted the experiment with fixed parameters, dataset size: 1M, number of filters = 2, the same query parameters except the one filter condition that triggers the analyzer to create siblings for a specific dimension. The values that we have tested for number of siblings are {1, 8, 10, 11, 14}. As we see (in Figures 4.4 and 4.5), both for the individual executions steps and for the total execution process, the increased number of siblings does not mean an increased execution time in general. This occurs because the analyzer operator creates one sibling query for every different dimension and not for every different value in the dimension, resulting in a quite stable execution time independently of the number of sibling queries.

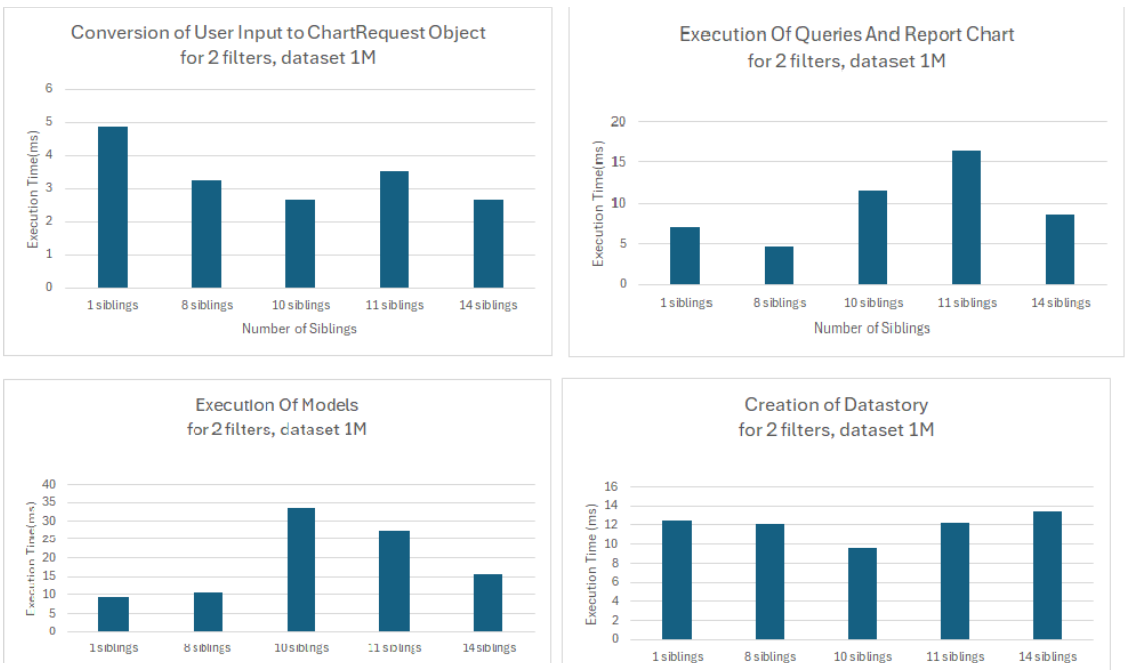


Figure 4.4 Execution time for different number of siblings.

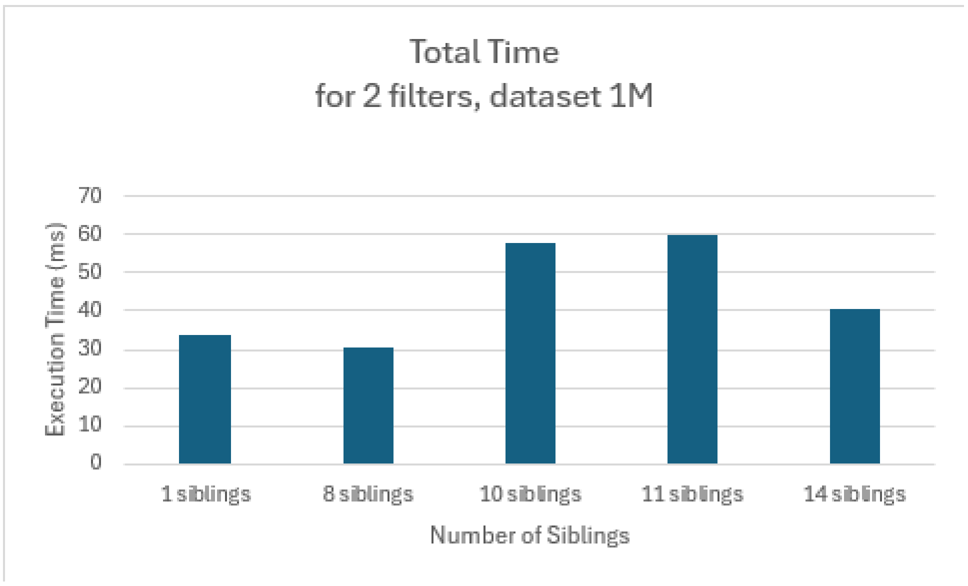


Figure 4.5 Total Execution time for different number of siblings.

CHAPTER 5

CONCLUSIONS

5.1 Summary of Results

The aim of this Diploma Thesis has been to extend an Business Intelligence system, specifically the Delian Cubes Engine, with a *Query-As-A-Chart* operator. The *Query-As-A-Chart operator* allows to construct a query not via the typical SQL syntax but in a friendlier way with the addition of specification of chart type. The result of the operator is (a) a set of charts that visualize the results, but most importantly, (b) a combined report in html, or *data story*. For the creation of the *Query-As-A-Chart operator*, two new packages were added to the Delian: ChartManagement and ChartRequestManagement responsible for the server and the client respectively. For the addition of new features in client (creation of query and display of charts), we used the JavaFX library extensively.

Upon completion of the operator implementation, an experimental evaluation was conducted as detailed in Chapter 4. This evaluation involved timing measurements aimed at assessing the impact of variables such as number of filters and number of siblings within the intentional query and dataset on the operator's execution time. In conclusion, the execution time is largely influenced by the time required for result completion and reporting, while extracting highlight models also play a significant role.

5.2 Future Work

The implementation of *Query-As-A-Chart operator* can be extensively optimized and enabled with new features, but most importantly to be corrected in ambiguous points.
First, for the time being, the system reads from serializable object the data for a unique time-series, while for multiple time-series, it reads the data from the mark-down file: File-ChartQueriesReport. In the future, the multiple series data can be transferred through the same serializable object.
Furthermore, the ranking of highlights is available only for unique-timeseries and not for multiple time-series and is something that can be implemented with a more careful look.
Simultaneously, there is the need for models like mega contributor to take into consideration the selected type of aggregation function of the chart query and change the calculation of score accordingly.
Finally, it would be nice in the future, the *Query-As-A-Chart operator* to enable visualizations for standard cube queries without the need for setting one time-dimension. Also, the operator should be able to create a suitable visualization type if the user does not specify a desired chart option.

References

[Apac99] The Apache Software foundation. Available at https://www.apache.org/ (1999).

[Bie13] T. D Bie: Subjective Interestingness in exploratory data mining. In: Tucker, A., Höppner, F., Siebes, A., Swift, S. (eds) Advances in Intelligent Data Analysis XII. IDA 2013. Lecture Notes in Computer Science, vol 8207. Springer, Berlin, Heidelberg (2013).

[DeCE18] P. Vassiliadis. Delian Cube Engine. Available at https://github.com/DAINTINESS-Group/DelianCubeEngine (2018).

[Dyre96] C. E. Dyreson. Information Retrieval from an Incomplete Data Cube. In Proc. 22nd Int. Conf. on Very Large Data Bases (VLDB), pp.532-543, Istanbul (Turkey), Morgan Kaufman Publishers, (1996).

[GeHa06] L. Geng, H.J. Hamilton: Interestingness measures for data mining: A survey. In ACM TODS, Association for Computing Machinery New York (USA), vol. 38(3), pp. 9-es, September (2006).

[GHG+22] S. Gathani, M. Hulsebos, J. Gale, P. Haas, C. Demiralp: Augmenting Decision Making via Interactive What-If Analysis. In 12th Annual Conference on Innovative Data Systems Research, (CIDR '22), Chaminade, USA, 9-12 January (2022).

[JePT10] C. S. Jensen, T. B. Pedersen, C. Thomesen. Multidimensional Databases and Data Warehousing. Morgan & Claypool, (2010).

[KiRo02] R. Kimball, M. Ross: The Data Warehouse Toolkit, 2nd Edition, John Wiley & Sons, Chichester, (2002).

[MaPA23] P. Marcel, V. Peralta, S. Amer-Yahia. Data Narration for the People: Challenges and Opportunities. In proceedings of the 26th International Conference on Extending Database Technology (EDBT), Ioannina, Greece 28th March-31st March, (2023).

[MaPV19] P. Marcel, V. Peralta, P. Vassiliadis. A Framework for Learning Cell Interestingness from Cube Explorations. In: Welzer, T., Eder, J., Podgorelec, V., Kamišalić Latifić, A. (eds) Advances in Databases and Information Systems (ADBIS). Lecture Notes in Computer Science, vol 11695. Springer, Cham, (2019).

[Mbaa21] O. Mbaabu. MOLAP vs ROLAP vs HOLAP in Online Analytical Processing (OLAP), Available at https://www.section.io/engineering-education/molap-vs-rolap-vs-holap/ , January 24 (2021).

[MDHZ21] P. Ma, R. Ding, S. Han, D. Zhang. MetaInsight: Automatic Discovery of Structured Knowledge for Exploratory Data Analysis. In Proceedings of the 2021 International Conference on Management of Data (SIGMOD), June 20-25, Virtual Event, China (2021).

[Ruso23] C. Ruso. KPIs vs Metrics: Learn the Difference with Examples From 2023, Available at https://www.datapad.io/blog/kpis-vs-metrics, March 25 (2023).

[Twin23] A. Twin. Key Performance Indicator (KPI): Definition, Types, and Examples, Available at https://www.investopedia.com/terms/k/kpi.asp, May 10 (2023).

[VaZi14] Alejandro Vaisman Esteban Zimányi Data Warehouse Systems Design and Implementation. Springer-Verlag Berlin Heidelberg (2014).

[Vass21] Panos Vassiliadis. Delian Cubes (as of 2021 01 29). Available at https://youtu.be/M4yulB2rHKQ?si=VQ1nWlO2H3-fN_sr, January 29 (2021)

[Yeol20] Y. Yeole. Data Storytelling-Basic Data Visualization in Excel, Available at https://yogeshyeole1111.medium.com/data-storytelling-basic-data-visualization-in-excel-8cd0ea52852a, October 5 (2020).

Short biographical sketch

Aggeliki Dougia was born on May 5, 1999, in Ioannina but grew up in Igoumenitsa. In September 2017, she was admitted to the Department of Computer Science and Engineering at the University of Ioannina and completed her studies in October 2022. Her diploma thesis was titled: Implementation of Evolutionary Optimization Methods for Association Rule Mining. In the same month, she began her studies in the master's program of the Department of Computer Science and Engineering, mainly choosing courses from the Advanced Computing Systems direction but and from Data Science and Engineering. Since November 2023, she has been working as a software engineer at P&I Hellas.