Departament de Llenguatges i Sistemes Informatics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Master in Computing

Master of Science Thesis

MULTIDIMENSIONAL QUERY RECOMMENDATION

Jovan Varga

Advisor: Dr. Patrick Marcel
Ponent: Dr. Alberto Abelló Gamazo

June, 23rd 2011

Acknowledgements

This thesis would not have been possible without my professors Patrick Marcel, Alberto Abello and Oscar Romero. My deepest gratitude goes to them for the support and guidance. Their patience and encouragement throughout the master thesis project have motivated me to enthusiastically work and constantly improve. I am honored and thankful for the opportunity to be their student.

It is a great pleasure to thank my family for being there for me always. Family values that I have gained directed me to the path of constant personal, professional and academic progress. My family has been supporting me in all mentioned aspects. I am very glad that I can share my success with them which makes it even more valuable. Nevertheless, I am truly indebted and grateful for their material support that enabled me to focus working on this thesis.

I thank also to my fellow students and friends with who I shared these precious student days. The friendships that we made now are the ones that last.

Index

Index of figures

Index of tables

1 Introduction

The main innovation in modern human society certainly is the presence and influence of Information Technologies. Everybody is somehow directed to use computers and different technological devices for either personal, academic, professional and other needs. Wide spread of Internet has opened totally new horizons. People are able to work from home, to communicate real time with others on the different sites in the world, to do their shopping online, to search for needed information on the web and many others. Social networks are used by people of all age, so boundaries in that aspect do not exist. These, extremely numerous, groups of Information Technology's users generate very large amounts of data.

Additionally, data are not generated only by human users that we mentioned above but also by various instruments and devices. For example, different types of sensors and sensor networks can be easily found in our everyday's surroundings. They measure light, movement, sound, pollution and many different facts that are needed for later processing and analysis.

Data are stored in database management systems (DBMSs). Data belong to various types and accordingly, database users that analyze them are different. Scientists manage and analyze large volume of experimental data. Economists and financial experts process financial data. Managers deal with data about sales of their products and efficiency of their employees. There are many more examples of diverse users that work over, not necessarily, distinct data. However, all of them tend to achieve the same goal of obtaining valuable and useful information out of large data volumes.

Powerful Business Intelligence tools enable us to work on this task. Data warehousing stores data from multiple information sources and transforms them into a common, multidimensional data model for efficient querying and analysis. Then, OLAP (On-line Analytical Processing) systems help us analyzing that data warehouse. OLAP is characterized by queries that are often very complex and involve aggregations. Due to the constantly and rapidly growth of data volumes, manipulation and analysis complexity also increases. In this situation, users, both expert and especially non-expert, would quite benefit from assistance with this task. This assistance can be achieved by analysis of multidimensional queries and their interconnection that comes from their belonging to same query session as it is explained it this thesis.

1.1 Motivation

Motivation for this master thesis subject comes from [1], an innovative paper on new important topics that the community will address in the next years, which was published in CIDR 2009 conference. There we have a thorough discussion of why analyzing and extracting information from DB query logs and query sessions is important. Traditional DBMSs provide sophisticated assistance for users in organizing, storing, managing, and retrieving data in databases, but at the same time capabilities for managing the already issued queries on data are limited. It can be said that they are relatively primitive. Usage of once tested and repeatedly re-used queries has become insufficient for more complex exploratory queries issued by advanced users, coming from both research and industrial areas. Such users could greatly benefit from more advanced query management tools.

Both, research and industrial, environments execute continuously changing queries. Those queries are logically connected between each other. As the desired analysis changes over time, so do the queries. Due to the increasing size of data sets, the cost of running a query has escalated and the traditional trial-and-error method for query development has become too expensive. In these settings, using knowledge obtained in previous queries by logging, organizing, and mining of the same can produce wealth of information that can assist users when querying the database. In this master thesis we focus on analyzing and extracting the information form multidimensional queries and query sessions with the goal of obtaining this knowledge.

Authors of [1] present a Collaborative Query Management System (CQMS) theoretical model targeted at large-scale, shared-data environments for browsing, searching and maintaining query log, and also recommending queries based on that log. Certain requirements that were raised with this model inspired research that led to this new approach, which is presented later. Paradigms raised for search in query log are:

- query-by-parse-tree, search for query conditioned by structure of the query
- query-by-data, search for query conditioned by query output
- query-by-feature, search for query conditioned by some query feature

For browsing the query log, a method for presenting query sessions instead of individual queries was suggested. These search and browse functionalities brought up research challenge of query representation and modeling as main focus of this thesis. Later, requirements for assisted interaction such as query completion, query correction and especially query recommendations brought up directions for future work related to the subject of this thesis.

1.2 Objectives

In this master thesis we will first summarize the recent efforts to support analytical tasks over relational sources. These efforts have pointed out the necessity to come up with flexible, powerful means for analyzing the issued queries and exploit them in decision oriented processes (such as query recommendation or similar). Issued queries should be decomposed, stored and manipulated in a dedicated subsystem. With this aim, we present a novel approach for representing SQL analytical queries in terms of a multidimensional algebra, which better characterizes the analytical efforts of the user. This thesis discusses how a SQL query can be formulated as a *multidimensional algebraic characterization*. Then, we discuss how to *normalize* them in order to *bridge* (i.e. collapse) several SQL queries into a single characterization (representing the analytical session), according to their logical connections. Afterwards, we talk about how this characterization can be exploited in a wide range of decisional tasks such as query recommendation and others. Finally, we present an implementation example of this approach with limiting it with *normalization* phase because it surprisingly turned out that it is hard enough to achieve with regards to time available. This implementation may later be upgraded to demonstrate the full potential of this novel approach.

1.3 Scope of the project

This document represents the master thesis, on the Master in Computing program, at Barcelona School of Informatics, Technical University of Catalonia. This master thesis is a research oriented. As so, we analyze the current situation in the field of multidimensional query recommendations and role of sessions of queries in it. Then we present one novel approach for analyzing queries in context of sessions of queries over decisional databases. These two contributions represent the theoretical part of the thesis. Afterward, this theory represents the basis of the practical part that follows it. Due to complexity and wide usability of this new approach as it is elaborated in this work, we cover only fundamental aspects of the new theoretical approach and present new perspectives for future research.

1.4 Organization of the Thesis

The rest of the document is organized as follows. Chapter 2 presents current State of the Art in the field of query recommendation with the discussion of important aspects that are lacking

and not dealt with. Chapter 3 introduces a novel approach which explains benefits of using Multidimensional Algebraic Structure of Analytical Queries. Chapter 4 discusses prototyping of this new approach, while Chapter 5 presents a demo example of the prototype. Chapter 6 shows and discusses the difference between initial schedule and final schedule with cost expectations and final costs. Finally, Chapter 7 states the conclusions and suggests guidelines for future work and research.

2 State of the Art

This chapter presents current State of the Art of, to the best of my knowledge, related approaches in field of query assistance and query recommendations. In all presented approaches special attention is brought to the relevance of the user sessions. Classification of the approaches is as follows. First classification is on Recommendations for relational databases and, on the other side, more specific case of Recommendations for data warehouses (multidimensional databases). Recommendations for relational databases will be further classified to the Recommendations after posting the query and Recommendations while writing the query. Recommendations for data warehouses will be sub classified to Methods exploiting a profile, Methods based on expectations, Methods exploiting query logs and Hybrid methods. Finally, the discussion about the importance of user sessions and ways of storing queries will be presented in this chapter since it is of crucial importance for the novel method that will follow afterwards.

2.1 Recommendations for relational databases

According to [2], recommender system is typically modeled with two sets, set of items, set of users from which we then generate item recommendations for the current user by using some function that takes both sets as an input and set of recommendations as an output. In [3] we find that these recommendations can be in general categorized into: (i) Content-based, that recommend items to the user similar to previous items highly rated by the same user, (ii) Collaborative, that recommend items highly rated by similar users and (iii) Hybrid, that combine content-based and collaborative ones.

Now, after this brief introduction about the recommender systems in general, we focus on recommendation systems in relational databases. In context of databases, items to recommend are queries. It has been just recently, in year 2009, that this application of recommendation systems has obtained the attention of research society. In this thesis, we categorize these approaches to Recommendations after posting the query and Recommendations while writing the query as earlier stated.

2.1.1 Recommendations after posting the query

As presented in [2] we can find only two attempts, [3] and [4], to formalize database query recommendations for exploration. These two approaches we assign to this category.

2.1.1.1 QueRIE: Query Recommendation for Interactive Exploration

In [3], the authors present conceptual framework and its instantiation for personalized query recommendations for interactive database exploration. When user explores the database during one session, he/she is usually searching for some specific information. Due to that relation the queries of one session are usually correlated. This observation represents a possible basis for generating recommendation for the next queries. These query recommendations can be suggested to all users that have similar querying interests. To generate these personalized query recommendations for current user this framework relies on both queries of past users and queries of current user so far. Proposed instantiation of framework is based on collaborative filtering.

When exploring the database user deals with the subsets of the database relevant to the analysis he/she wants to perform. In this framework it is assumed that this subset is modeled as a session summary. Level of details captured by session summary can be different. On one extreme, session summary can contain only the names of the relations that appear in the queries of the user, and the number of queries that reference it as a way of measuring its importance. On the other, detailed session summary may contain the actual results inspected by the user, along with an explicit rating of each result tuple.

For generating recommendations, this framework first constructs the summary for each user, then generates a "predicted" summary for the users and finally based on "predicted" summary generates recommendation queries for the user.

Proposed instantiation of the framework is a Witness-Based Collaborative Filtering approach. In general, this instantiation models session summaries as a weight vector of witnesses. Witness is a tuple of the database that contributes in at least one answer to at least one query of the session. Weight of the vector element (witness) represents an importance of it for a session. It can be computed in two ways, either by binary weighting schema – a witness participates in query answer or not, either by result weighting schema – a witness's weight is reciprocal of number of tuples in query answer (more answers, less weight and vice versa). For generating of "predicted" summary instantiation uses similarity function between two sessions (realized with any vector-based metric) and "mixing" factor between 0 and 1 that determines relation which users' traces will be taken more into account, current user's or past users'. For generating recommended queries, query vectors (weight vector of tuples for a query) are compared with

"predicted" summary, and then past user queries which are expected to be easily human understandable are ranked according to "predicted" summary.

The experimental evaluation of the proposed instantiation is also presented in the paper. The authors provide QueRIE (Query Recommendations for Interactive data Exploration) architecture for information flow in their approach, shown in Figure 1.

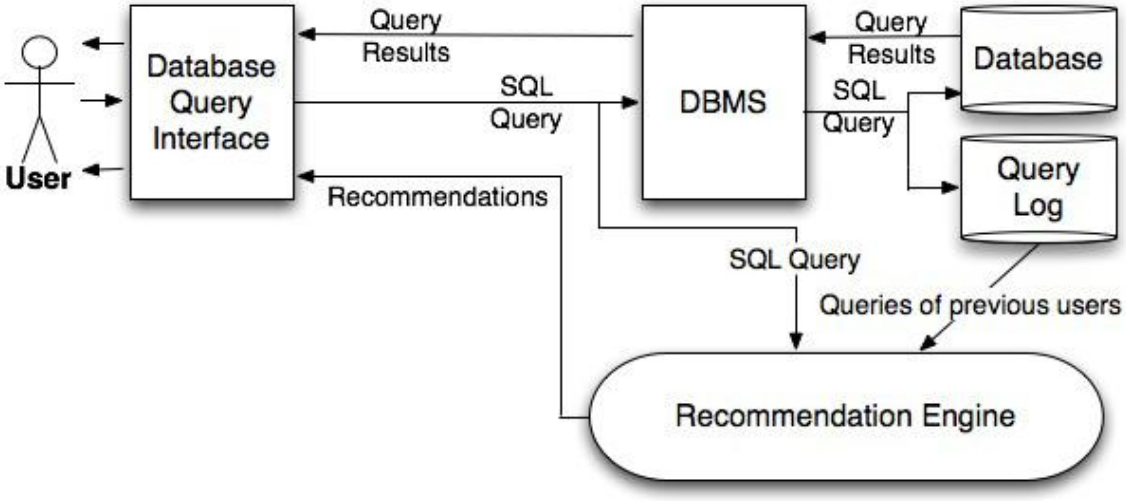

Figure 1. QueRIE Architecture, figure taken from [3]

The active user's queries are forwarded to both DBMS and the Recommendation Engine. The DBMS processes each query and returns a set of results. At the same time, the query is stored in the Query Log. The Recommendation Engine combines the current user's input with information gathered from the database interactions of past users, as recorded in the Query Log, and generates a set of query recommendations on above explained way and returns them to the user.

Related to [3], in [5] authors present improved QueRIE system. We find new, alternative "fragment-based" recommendation engine besides existing "tuple-based" recommendation engine using the same recommendation methodology. This new recommendation engine is needed because other "tuple-based" recommendation engine is not able to detect when two queries are semantically similar but retrieve different results due to some filtering conditions. The "fragment-based" recommendation engine deconstructs each query into fragments and discovers other fragments that co-appear with them in sessions of different users (an indication of structural similarity). Then these fragments are used to identify the most similar queries and generate the final recommendation set.

As in [3], framework consists of three components: the construction of session summary for each user, the computation of a "predicted" summary for active user, based on the active user's and the past users' summaries, and the generation of queries based on "predicted" summary. Now we focus on specificities of "fragment-based" recommendations.

The approach for "fragment-based" recommendations is based on the pair-wais similarity of query fragments (attributes, tables, joins and predicates). For generating recommendations we need to identify fragments that co-appear in several queries posed by different users. The session summary vector for a user consists of all the query fragments of the past user's queries. Every vector element (query fragment) is weighted, either binary, either by weighting schema, analog as in "tuple-based" engine. For computing the "predicated" summary, fragment-fragment matrix that contains all similarities is constructed using the session summaries of the past users and a vector similarity metric. Then "predicated" summary represents the estimated importance of each query fragment with regard to the active user's behavior. For generating recommendations, top-*n* fragments are selected based on "predicted" summary. Then all past queries are ranked based on a normalized metric measuring the number of common query fragments of each query to the top-*n* list.

Let us now summarize representation of the queries and query sessions in this QueRIE approach. In case of "tuple-based" recommendation engine we have user-user collaborative recommendations where the queries suggested to the user are the ones that retrieve similar tuples posed by other users. When using binary weighting schema, a query is represented as a binary vector with the length of the number of tuple in the database instance, where one element has value 1 if the query used the tuple and 0 otherwise. A session is also represented by a binary vector with length of the number of tuples in the database instance, where one element has value 1 if at least one query of the session used the tuple and 0 otherwise. Example of this storing of sessions is illustrated in Table 1.

Table 1. QueRIE: Tuple-based, binary weighting schema, session's summary

	Tuple 1	**Tuple 2**	**Tuple 3**	**…**	**Tuple n**
Session 1	1	0	0		0
Session 2	0	1	1		1
Session 3	0	0	0		1
…					
Session m	1	1	0		0

In this setting, if we have a current session $S_c = (1,...,0)$, we compare all other sessions $S_i, 1 \le i \le m$ with S_c and recommend S_i that is the closest to S_c.

Similar, in case of "fragment-based" recommendation engine we have fragment-fragment collaborative recommendations where the queries suggested to the user are the ones posed by other users in which fragments of the current query co-appear. When using binary weighting schema, a query is represented as a binary vector with the length of the number of fragments in the log, where one element has value 1 if the fragment appears in the query and 0 otherwise. A session is also represented by a binary vector with length of the number of fragments in the log, where one element has value 1 if the fragment appears in at least one query of the session and 0 otherwise. Example of this storing of sessions is illustrated in Table 2.

Table 2. QueRIE: Fragment-based, binary weighting schema, session's summary

	Movies	**Actor=value**	**Genre**	**…**	**Price**
Session 1	1	0	0		0
Session 2	0	1	1		1
Session 3	0	0	0		1
…					
Session m	1	1	0		0

In this setting, if we have a current session $S_c = (1,...,0)$, we compare all other sessions $S_i, 1 \le i \le m$ with S_c and recommend S_i that is the closest to S_c.

Discussion about this, and other, representation of the queries and query sessions we have at the end of this chapter.

2.1.1.2 YMAL

One more approach for generating query recommendations for the users we find in [4]. With results of each query, the user gets additional recommended results that are called "You May Also Like" or YMAL results. These additional recommended results might be of user's potential interests.

This framework works with traditional select-project-join (SPJ) queries, over a database system for a set of users. Importance of a query for the user is measured by the number of times the user posed that same query. Approaches to generate YMAL results consider are:

- *Current-state* approaches, which exploit the content and schema of the current query result and database instance. Focus in the paper is on this approach and we discuss about it more detailed later.
- *History-based* approaches, which exploit the history of previously submitted queries to the database system, e.g. by using query logs. Here the utility of a query for the user is equal to the number of times the user has posed that query. Recommendations generated can be either *query-based* YMAL results (similar to *content-based* recommendations), either *user-based* YMAL results (similar to *collaborative* recommendations).
- *External sources* approaches, which exploit resources external to the database, such as related published results and reports, relevant web pages, thesaurus or ontologies.

This categorization is illustrated in Figure 2.

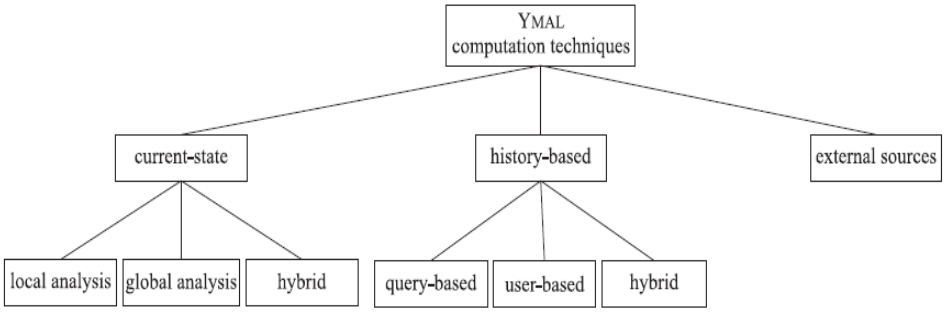

Figure 2. A taxonomy of YMAL computation techniques, taken from [4]

With focus on *current-state* approach, authors present methods for computing YMAL results. This approach is content-based. We have two kinds of analysis methods belonging to this approach that are in detail presented, local and global ones.

Local analysis aims at discovering interesting patterns that are later recommended YMAL results. The subject of analysis is either result of query i.e. tuples retrieved, either expanded result of query i.e. extended tuples produced by joining previous tuples with other tuples in the database.

When analyzing the result of a query, method discovers frequent values of certain attribute and based on that value recommends other tuples that are not already selected but have the same values of that attribute. For example, if we select the title and genre of all movies of Chris Lee, we discover that the result contains mostly movies of "fantastic" genre, than we can select other movies of that genre not already selected.

In case of analysis of expanded result of query, situation is similar. We first expand query by joining with additional relations in the database. Then we discover some frequent values of

certain attribute and than by analysis of schema we notice other values that often go together with first discovered value and based on that generate recommendations. Continuing the previous example, if we select title and genre of the movies of Chris Lee, by analyzing schema we expand this query to show also Production Company of movies. Then we discover that he often plays in movies produced by Paramount Pictures and we can recommend some other movies of this Production Company.

Global analysis aims at discovering correlations among attribute values or relations. For this purpose certain statistics for database needs to be maintained. Based on this database statistics pairs of attribute values or pairs of relations that most frequent go together are discovered and recommended. Continuing the previous example, if we select title and genre of the movies of Chris Lee, by analysis we discover that he often stars with Jet Lee (pairs of attribute values). Then we can suggest movies in which Jet Lee plays. In case of pairs of relations, if the relation that contains movie title is often connected with relation containing movie director name, than we can join that two tables and present also the names of movie directors of all movies presented along with the title and the genre of the movie.

These examples are simple illustrations, just to give us a general picture. We do not go into all the details of the method described because for the analysis of query and query sessions representations it is not needed.

As we have seen in this approach, queries are stored just as they are and they are analyzed by the tuples that they retrieve. Query sessions are not mentioned because authors put focus on methods for *current-state* approaches and do not mention any methods for history-based approaches where we would expect to have query sessions.

2.1.2 Recommendations while writing the query

Recommendations while writing the query generally focus on query fragments and assist user to finish writing query by suggesting next possible fragment. We find two attempts of this kind, [6] and [7].

2.1.2.1 SnipSuggest

In [6], we find an approach of recommendation for autocompletion of query named SnipSuggest: Context-Aware Autorcompletion for SQL. SnipSuggest system architecture is shown in Figure 3. It consists of Query Logger and Query Eliminator that work over Query Repository. Query Repository represents query log. Query Logger logs queries while also

extracting various features from them. Feature can be predicate in the WHERE clause, column name in SELECT clause and similar. Query Eliminator periodically analyzes the most recent queries and drops some of them. It processes query sessions with the goal to reduce workload size, and the recommendation time, while maintaining recommendation quality. In this approach, a query session is a sequence of queries written by the same user as part of a single task. The Query Eliminator eliminates all queries, except those that appear at the end of a session.

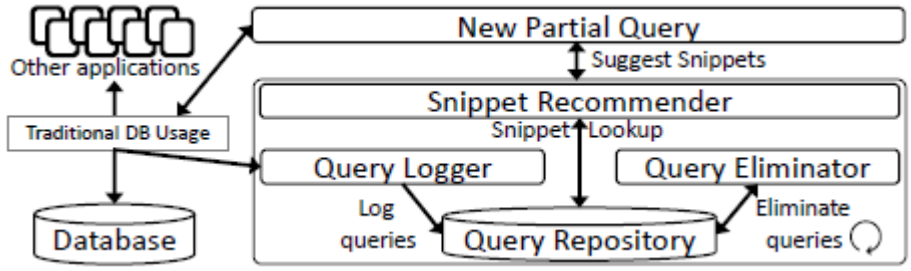

Figure 3. SnipSuggest System Architecture, taken from [6]

SnipSuggest enables user when composing the query, to select clause and ask for recommendations for that clause at any time. SnipSuggest's goal is to recommend k features that are most likely to appear in that clause in the user's intended query. System views the space of queries as a directed acyclic graph (DAG). It models each query as a set of features and every possible set of features becomes a vertex in the DAG. When asked for recommendation, SnipSuggest transforms the user's partially written query into a set of features, which maps onto a node in the DAG. Each edge in the DAG represents the addition of a feature so the recommendation problem translates to that of ranking the outgoing edges for the vertex that corresponds to the user's partially written query. All queries in the Query Repository that are descendants of the current vertex in the DAG are referred as the potential goals for the partially written query. An example, if the query log contains two queries:

- SELECT title, genre FROM Movies WHERE actor = 'C. Lee'
- SELECT title FROM Movies WHERE director = 'Allen'

DAG would look like the one showed in Figure 4.

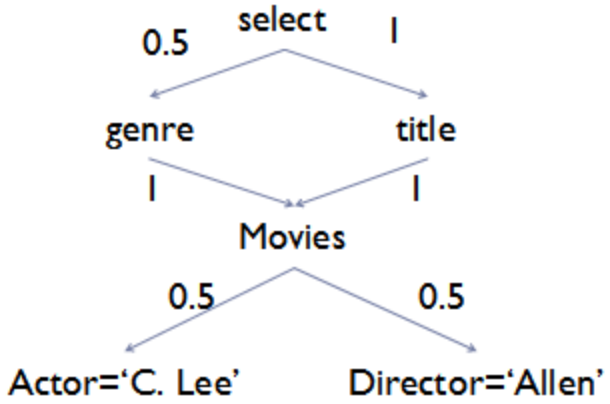

Figure 4. Example of SnipSuggest DAG

SnipSuggest uses two algorithms to recommend suggestions based on the current context: SSAccuracy and SSCoverage. SSAccuracy, for a partial query q and a query workload W, suggests the k features with the highest conditional probabilities given q. If q's features have appeared together in past queries, SSAccuracy is able to efficiently identify the possible additional features with the highest probabilities. SSCoverage as another option tends to suggest the k features that maximize the probability that at least one suggestion is helpful. The goal is to diversify the suggestions, to avoid making suggestions that all lead toward the same query. It turns out that the problem is NP-hard so the authors (of [6]) propose an approximation algorithm.

To summarize, this is collaborative approach that does on the fly assistance to users writing complex queries. Assistance is done, on request, by generating recommendations for current query fragment based on the analysis of query log. Analysis implies the construction of a graph where nodes are the sets of fragments appearing in queries. Edges indicate the precedence and take value in interval [0, 1]. Partially written query is indentified as a node in the graph and the recommendations are node's successors. So, in this approach query is modeled as a node in the graph containing the set of query features. A query session is modeled by the ultimate query posed because it is considered that it covers all the previous queries in the session.

2.1.2.2 Recommending Join Queries Via Query Log Analysis

One more approach for helping users with completing the complex queries is presented in [7]. Its goal is automatically create join query recommendations based on input and output specifications. This specification consists of:

- the input (or start) tables on which conditions of the form "*Table.atribute = value*" are present in the SQL WHERE clause,

- the output (or end) tables whose attribute values are in the result of the query, i.e. in the SQL SELECT clause.

The recommended join query includes:

- intermediate tables that are added to the SQL FROM clause,
- join conditions that are added to the SQL WHERE clause, and connect the input and output tables via the intermediate tables.

This method analysis existing query log and extracts joins made in the previous queries. Then those joins are used for generating recommendations to the current user. Obliviously, method uses collaborative approach.

Example of this recommendation method, if the query log is:

- *SELECT title, budget FROM Movies NATURAL JOIN Production WHERE director='Allen'*
- *SELECT title, budget FROM Movies NATURAL JOIN Production WHERE director='Allen' and year > 2000*
- *SELECT title, budget, year FROM Movies NATURAL JOIN BoxOffice WHERE director='Allen'*

And if the current query begins with:

- *SELECT title, budget FROM Movies*

Then system should recommend: *Movies NATURAL JOIN Production*

All the queries from the log are used to create general schema graph where the nodes represent tables, while edges are joins between tables. A singe query in this query corresponds to a connected sub graph in the schema graph. From the nodes that represent tables appearing in the WHERE clause, through joins with intermediate table(s) method reaches the tables appearing in the SELECT clause. As we can see, only information extracted about the query are the joins. User sessions are not mentioned neither modeled.

2.2 Recommendations for data warehouses

In this section, we overview the kinds of approaches related to query recommendations for data warehouses (multidimensional databases). We take advantage of survey paper [2] where we can find classification and analysis of this subject.

Basic peculiarities of typical data warehouse exploration found in [2] and important for further analysis are:

- A data warehouse is a read-mostly database and data are added, never or very seldom deleted. In this settings, it is quite common that a user issues the same sequence of queries more than once, changed only in some parameter e.g. from one year to another.
- A data warehouse is a database shared by multiple users. Users' interest may vary over time so it would be important to issue recommendations computed from other users' habits and at the same time respecting the user interests and privacy.
- A data warehouse has a particular schema that reflects a known topology (multidimensional model), often called the lattice of cuboids, which is systematically used for navigation.
- A typical analysis session over a data warehouse is a sequence of queries that has a sense w.r.t. some expectations. Sessions (as sequences of queries) are of particular importance in this context since by this sequence the user navigates to discover valuable insight w.r.t. her expectations or assumptions.

Important concepts for further analysis of data warehouse query recommendation:

- A data warehouse query (or query for short) – any query expressible in a given language for manipulating multidimensional data
- A session – sequence of queries
- A log – a set of sessions
- A data warehouse instance – a set of $n+1$ relation instances where n instances play the role of dimensions and one instance is the fact table
- A user profile – any information allowing definition of an order over the tuples in the fact table
- An expectation function – any function on a data warehouse instance that produces a score (often a real number).

By using these terms we formalize query recommendations for data warehouse exploration as a function *Recommend*(*L*,*cs*,*I*,*P*,*f*) that has the following parameters:

- *L*: A set of sessions (the query log),
- *cs*: A particular session (the current session),
- *I*: A datawarehouse instance,
- *P*: A user profile,
- *f*: An expectation function.

These parameters allow to cover Current-state (*I* and *f*), History-based (*L* and *cs*), and External sources (*P*) approaches found in [4].

The function *Recommend* returns an ordered set of pairs as recommendations, each pair composed of a query with its associated rating indicating the relevance of the query for the current session.

With this formalization, an explicit user rating for queries is not assumed. It is only considered that once a query is part of a session, it is relevant for the session. Note that the queries that do not appear in the former sessions (the log) neither in the current session may be constructed by *Recommend*.

In the next sections, according to [2] we present the categories of approaches for data warehouse query recommendations by parameters they use for calling *Recommend* function. There are four approaches in this classification: Methods exploiting a profile, Methods based on expectations, methods exploiting query logs and Hybrid methods.

2.2.1 Methods exploiting a profile

This category includes works that suppose that a profile is provided together with the current session. A profile expresses user preferences over the tuples of the fact table. Methods in this category do not consider any expectations neither log, but take the data warehouse instance into account. The call to the *Recommend* function for this category is the following: *Recommend*(-, *cs*, *I*, *P*, -), where *cs* is the current session, *I* is a data warehouse instance and *P* is a profile.

For all these methods, the profile *P* and the instance *I* are used to modify the current query (i.e. the last query of *cs*). The queries that are the result of this modification constitute the recommendation.

The advantage of the methods in this approach is that recommendations are computed w.r.t. both a profile and the user sessions. Thus different users will obtain different recommendations.

2.2.2 Methods based on expectations

This category includes works that rely on discovery driven analysis, where a model on unseen data is used together with the already seen data, i.e. the results of the launched queries. The strongest deviations to the model are recommended.

Recommendation methods based on discovery driven analysis consist in recommending queries that result in data that most deviate from a model (that we call expectation). The call to the *Recommend* function for this category is the following: *Recommend*(-, *cs*, *I*, -, *f*) where *cs* is the current session, *I* is the data warehouse instance and *f* is an expectation function. Among the works in this category, the main difference is the model used, i.e., the nature of the function *f*.

2.2.3 Methods exploiting query logs

This category includes works that suppose that a query log is used to look for similarities between the current session and former sessions, to extract one query as the recommendation. The call to the *Recommend* function for this category is the following: *Recommend*(*L*, *cs*, *I*, -, -) where *L* is a query log, *cs* is the current session and *I* is the data warehouse instance. Among the works in this category mainly difference is the way they consider the similarity between sessions and/or queries.

2.2.4 Hybrid methods

This category includes the works that combine some or all the previous methods. According to [2], for now there is only one work using hybrid method. Because of the nature of this category the call to the *Recommend* function varies and it can be represented with the following: *Recommend*(?, *cs*, *I*, ?, ?) where *cs* is the current session, *I* is the data warehouse instance, and the ? may be "-" (empty) or *L* (query log), "-" or *P* (user profile), "-" or *f* (expectation function) respectively.

2.3 Discussion

In previous sections we have reviewed the current approaches for query recommendation for relational databases and their ways of storing and managing queries and query session. Also, we reviewed the importance of query sessions for data warehouses followed by different kinds of methods for query recommendations over data warehouses. As the data warehouse is a database with special settings we can easily deduct that we can have the data warehouse on the top of a relational database (as well as decisional, not multidimensional databases can be implemented using the relational technology). This assumption is the starting point used in next chapters to present a novel approach for analyzing queries over decisional databases. Main focus in this

master thesis is on identifying and normalizing Multidimensional Algebraic Structure (MAC) of analytical queries which happened to be surprisingly hard enough and relevant since it is a preliminary step for query recommendation and other uses. Detailed explanation of this process is presented later. For now we just point out that query recommendation hugely relies on the structured way we store queries and query sessions.

A query session in analytical environments is typically characterized by mutual correlation between queries that constitute it. This comes from the fact that a user who navigates over data usually tries to discover some useful information. In order to do so, normally the user modifies the posted query until he/she reaches the goal. So, every query posed by that user in one query session is usually a modification of a previous query, except for the first query of course. This is also reflected on the query content and the structure of the query and that is what we want to detect and exploit in multiple ways. Storage and management of queries must represent user query sessions in such a way that it contains information about queries posed, order between queries of the session and user's behavior in navigating data. Possibilities for exploiting information about queries in the form of query recommendation and similar usages are directly influenced by the way we store the query sessions and corresponding queries. One example of this fact has been shown in Section 2.2 where we have reviewed categories of methods for generating query recommendations. All represented methods use current session as an input and therefore they are directly influenced and dependent from the way of storing that information.

Now, when stated why the query sessions are important we analyze existing ways for handling queries in reviewed approaches for relational databases. The accent is on this kind of approaches since this master thesis focuses on relational data sources. We characterize these approaches consistently with the motivation presented in the Section 1.1. The deficiencies of the approaches correspond to the paradigm they belong to. Categorization of the approaches:

- In the case of QueRIE, presented in Section 2.1.1.1, we noticed that a query is represented as a vector marking touched tuples or contained query fragment, and thus it belong to **query-by-data** or **query-by-parse-tree** paradigm respectively. Session is similarly presented as a vector marking touched tuples in the session or query fragments contained in at least one query of the session.
- In YMAL approach, presented in Section 2.1.1.2, query session is not even stored while the query is stored as it is, and characterized by the tuples it retrieves. Obviously this approach uses **query-by-data** paradigm.
- In the case of SnipSuggest, presented in Section 2.1.2.1, query is stored as a node in the graph containing the set of query features. That means that queries are analyzed

simple, directly by analysis of does query contains some fragment or not. Hence used paradigm is **query-by-parse-tree**. Session is kept as a last query in the session log so the interconnection between all other queries that belong to the same session is lost.

- Last approach, presented in Section 2.1.2.2, even does not store all the information about the query but only about the joins contained in queries. This way no other information, except for recommending joins, can be obtained from query log. **Query-by-parse-tree** seems to be covering paradigm for this approach. Sessions do not even make sense in this context.

From previous categorization it is clear that current approaches belong either to **query-by-parse-tree** either **query-by-data** paradigms. Although these paradigms have some useful characteristics we highlight the deficiencies that caused the need for better and more powerful approach.

Query-by-parse-tree that focuses on syntactical structure of the query and this way it requires that everything recommended has to already appear in the log and be recorded at some point in the past. This is not convenient for the explorations when the user is searching for something unexpected and not discovered yet. Also, for all the recommendations we only know that they syntactically suit but we do not know if they semantically make sense which is big deficiency. This paradigm does not keep the semantic characterization and interconnection between the queries and in such a way it loses important information about the queries and query sessions that could be very valuable. Sometimes queries with same effect are differently formulated and this would not be identified. Query-by-parse-tree cannot detect granularity dependency between the queries (in one query session) on the semantic level and as such apply it to some other not queried or not related searches. In short, it only detects syntactical concepts in the query log and uses them to recommend queries composed of appropriate query parts with highest ratings.

Query-by-data paradigm seems alternative option from the one previously discussed. It focuses on query outputs and as such entails disadvantage of not recognizing two similar queries that due to some filtering condition retrieve different data. This paradigm implies similarity based on data match only without any additional semantics such as interconnection between the queries and query sessions that manifest similar navigation over data. Just focusing on data instances has many shortcomings in the context of data warehouses where we have enormous amounts of data and it is questionable how efficient approaches using this paradigm could even be for the interactive real time assistance.

It seems that neither solution by now is powerful and flexible enough for some advanced exploitation. For the purpose of storing the information about multidimensional queries, the semantics those queries contain and query sessions it is obvious that we cannot use any of the previous discussed ways. We have to come up with better, more meaningful solution. In our approach, this task is tackled with the novel approach that uses the **query-by-feature** paradigm since it extracts semantics of multidimensional operations and in such a way it is not directly related to either the syntactical characteristics of the query, or data retrieved by the query. This extracted semantic can be represented in various syntactical ways and also when applied to the different concepts it can retrieve various data instances. We focus on this novel approach in next chapter where it is presented in detail.

3 Multidimensional Algebraic Structure of Analytical Queries

This chapter presents the novel approach for representing SQL analytical queries in terms of a multidimensional algebra, which better characterizes the analytical efforts of the user, developed by my professors Patrick Marcel, Alberto Abello and Oscar Romero [17]. I am greatly honored that I had an opportunity to participate and give my contribution in this research project. The next chapter gives an overview of the practical part of my work while here I present theoretical basis and the developed framework.

As we saw, recent efforts to support analytical tasks over relational sources have pointed out the necessity to come up with flexible, powerful means for analyzing the issued queries and exploit them in decision-oriented processes (such as query recommendation). Issued queries should be decomposed, stored and manipulated in a dedicated subsystem.

This framework provides a new solution for analyzing the issued queries and storing them in a structured way that facilitates their reuse and exploitation. Exploitation considers understanding semantics of the query, comparing the queries, clustering the queries into groups and other means that enable us future tasks such as query recommendation or physical and conceptual design. For storing queries in a structured way, some kind of smart, *normalized form* is required. In this sense, the relational algebra would be a candidate to characterize the input queries. However, this novel approach moves a step further as it has already been discussed that the *whole* relational algebra does not properly suit for analytical queries [8]. By analytical queries we assume all the queries that can be represented from the multidimensional point of view. Therefore it is proposed the use of a *multidimensional algebra*. The correspondence between both algebras have already been studied in the literature and it has been shown that the *multidimensional algebra* is a subset of the relational one [9]. Note that this is sound with the discussion introduced in [8] where it is said that the extended relational algebra is, simply, too expressive (in the sense it provides functionalities not needed) from an analytical point of view. Thus, the *multidimensional algebra* is simpler, and we can use it to express the analytical efforts of the user in a more concise, effective way.

3.1 Terminology and Notation

Before presenting the new approach we first need to explain and clarify basic concepts and the terminology used. We start from the general technological environment and concepts related to it and then focus on the multidimensional algebra and terms related to the new framework.

As earlier stated this framework is based on OLAP. By OLAP (On-line Analytical Processing), according to [10], we consider an approach to decision support, which aims to extract knowledge from a data warehouse and its main idea is providing navigation through data to non-expert users, so that they are able to interactively generate ad hoc queries without intervention of IT professionals. Name OLAP was introduced in contrast to OLTP (on-line transactional processing), so that it reflects the different requirements and characteristics between these classes of uses. Characteristics of OLAP, along with differences in respect to OLTP are presented in Table 3.

Table 3. Comparing OLTP versus OLAP, according to [10]

	OLTP	OLAP
Usage	Application specific	Decision support
Workload	Predefined	Unforeseeable
Access	Read/Write	Read-only
Query structure	Simple	Complex
Records per operation	Tens/Hundreds	Thousands/Millions
Number of users	Thousands/Millions	Tens/Hundreds

The main characteristic of OLAP is multidimensionality. The **data cube** metaphor, or just C**ube** in this document, is used to make user interaction easier and closer to decision makers' (usually managers, directors and similar) way of thinking, who would probably find SQL or any other text-based query language hard to understand and error prone. Thus, it is much easier for them to think in terms of the multidimensional model, where a Fact is a subject of analysis and its Dimensions are the different points of view that analysts could use to study the Fact. In this way, the instances of a Fact are shown in an *n*-dimensional space usually called **Cube**. Example of a Cube we have in Figure 5, where the Dimensions are the axes of the cube, while Fact is each cell of the Cube.

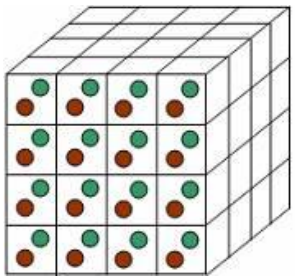

Figure 5. Data cube

Multidimensionality is based on the **fact** / **dimension** dichotomy. The **fact** or subject of analysis is placed in the *n*-dimensional space produced by the analysis dimensions. We consider a **dimension** to contain an aggregation hierarchy of **levels** representing different granularities (or levels of detail) to study data, and a level to contain **descriptors** (i.e. level attributes). We differentiate between identifier descriptors (univocally identifying each instance of a level) and non-identifier. In turn, a **fact** contains analysis indicators known as **measures**. A level of detail for each dimension produces a certain data granularity or **data cube**, in which place the measures. Dimensions with its descriptors together with fact containing measures and slicers represent the **schema of the data cube**. Finally, we denote by **base** of the space a minimal set of levels identifying univocally a certain data granularity. Figure 6 illustrates one example of a multidimensional schema. We have three *dimensions* – Date, Product and Location and one *fact* table – Sales. Dimensions have intuitive hierarchy of *levels*, for example Date has a day/month/year hierarchy. In dimension Product we find both *identifier descriptors* – product/category and *non-identifier descriptors* – description. Fact table Sales contains two *measures* – price and discount. Data cube for this schema can be represented by Figure 5, where axes are Date, Product and Time, and cells represent one instance of Sales.

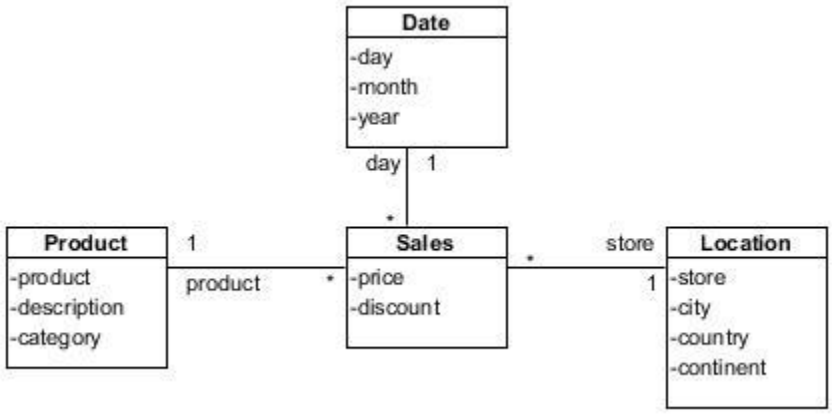

Figure 6. Simple multidimensional schema

3.1.1 The Multidimensional algebra

Multidimensional Algebra (MDA) presented in [10], was proven to be closed, complete (regarding the cube-query in [8]) and minimal (see [11]), and consists of the operators shown in Figure 7 (colored dots and triangles represent measures in a cell, for grasping their intuition) and explained below. All the operators are unary (i.e. apply within a cube) except for drill-across and set operators, which operate over two cubes.

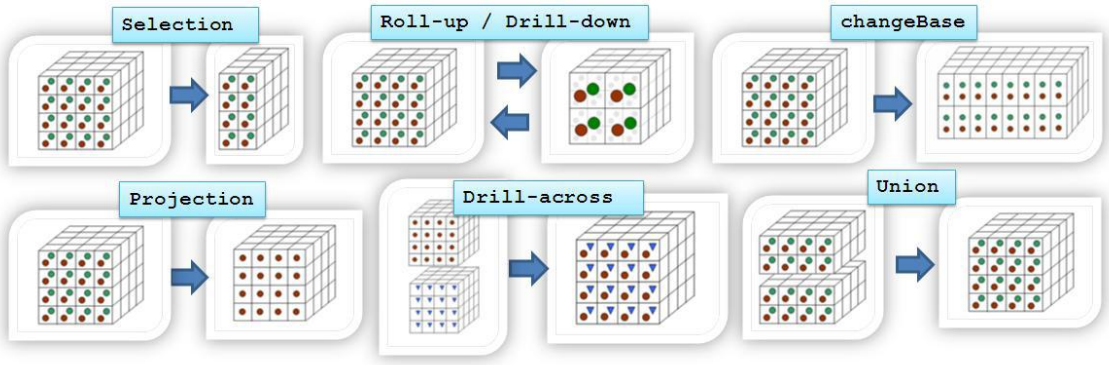

Figure 7. Conceptual exemplification of the MDA operators

Descriptions of multidimensional operators:

- **Selection** ($\sigma_p cube$): By means of a logic predicate *p* compound of clauses over descriptors, this operation allows to choose the subset of points of interest out of the whole *n*-dimensional space. In the case of multidimensional schema shown in Figure 6, example of selection would be if for dimension Location we have *city* = "Barcelona".
- **Roll-up** ($\gamma^{level_i \to level_j}_{f(measure_1),\ldots,f(measure_n)} cube$): This operation groups data instances in the cube based on an aggregation hierarchy. This operation modifies the granularity of data by means of a many-to-one relationship which relates instances of two levels in the same dimension, corresponding to a part-whole relationship. Drill-down (i.e. the counterpart of roll-up), can only be applied if previously roll-up was performed and the correspondences between instances was not lost. Again, in case of the multidimensional schema shown in Figure 6, example would be if we move from dimension Location level city to level country and aggregate the measures (price, discount) of fact table Sales.
- **Projection** ($\pi_{measure_1,\ldots,measure_n} cube$): It selects a subset of measures. Continuing the previous example, projection is selecting from fact table and showing only *price* measure (and leaving out the others).

- **ChangeBase** ($\chi_{base_1 \to base_2} cube$): This operation reallocates exactly the same instances of a cube into a new *n*-dimensional space with exactly the same number of points, by means of a one-to-one relationship. Actually, it allows replacing the current base by one of the alternatives, if more than one set of dimensions identifying the data instances (i.e. alternative bases) exist. To illustrate effect of this operation, we shall assume that each product is produced by only one producer and that category of products maps to group of producers. Then we can exchange the basis of a cube from Product, Date, Location to Producer, Date, Location.
- **Drill-across** ($cube_1 \bowtie cube_2$): This operation fuses the measures in two cubes related by means of a one-to-one relationship. The n-dimensional space remains exactly the same, only the instances placed on it change. For example, continuing example shown in Figure 6, if we would have one more cube with fact table Ratings with measure rating, over the same dimensional space (Product, Date, Location) then drill-across operation between these two cubes would produce a new cube with same dimensions over fact table containing all measures – price, discount, rating.
- **Set Operations** ($cube_1 \theta cube_2$): These operations allow operating with two cubes if both are defined over the same *n*-dimensional space. We consider union ($\cup$), difference (\) and intersection ($\cap$). Nevertheless, from here on we focus on union, since the same considerations can be applied to the others. Also, we assume a perfect data cleaning and ETL phase (if the same cell appears in two different cubes, measures coincide). On our running example (Figure 6), if we would have two cube over same dimensional space, one with selection over Location dimension *city* = "Barcelona" and other with selection over Location dimension *city* = "Madrid", union would produce one cube over same dimensional space containing both values for the level city of dimension Location.

Each of these MD operators has its MD operator schema. Operator schema represents either clause over a descriptor (slicer), either descriptors and/or measures that define the effect of the operator. Furthermore, we denote by cube schema the output schema resulting of applying each operator schema over the query raw data (see Section 3.1 for the formal definition of data cube schema). The output data cube schema usually contains modified set of descriptors and measures (except for example in the case of selection where this set is not changed).

3.1.2 Terms and basic concepts of novel approach

Now we introduce some basic terms and concepts specific for the novel approach. All of them are later more detailed explained in case something is not clear enough in this section.

In this approach, it is proposed to characterize each issued SQL query (i.e. each query in the query log) by means of the set of MDA operators mentioned and presented earlier. Details and examples are presented later.

Each operation of this algebraic characterization is associated with corresponding multidimensional schema. This means that after each multidimensional algebraic operation over multidimensional schema we know which attributes (either factual or dimensional) are in the output, the result. However, this characterization is not intended to be executed but to keep track of the knowledge captured in analytical queries form a multidimensional point of view. Indeed, it is a characterization giving multidimensional sense to the query and that is what we call ***Multidimensional Algebraic Characterization*** (**MAC** from now on). MAC forms a ***tree*** (like in the relational algebra, due to binary operators such as union or drill-across, as we in detail explain later). Leafs of this *tree* are tuples *directly* retrieved from the database (i.e. the materialized data) and thus, we refer to them as ***raw data***. Other nodes in this tree represent multidimensional operations (manipulations) over *raw data*.

Once the query has been characterized according to multidimensional algebra the next step aims at *normalizing* the MAC. Objective of ***normalization*** is to facilitate future manipulation and comparison. To do so, it is compulsory to store each MAC in a *normalized* form. In this new approach, we benefit again from the algebraic structure proposed, and we use a set of *equivalence rules* (based on those of the relational algebra) to pull the multidimensional operators up the algebraic structure. The final product is ***Normalized MAC*** (***NMAC*** from now on).

After we normalize query and generated NMAC we can use this standardized form for gaining further information and semantics. By saying that, it is considered that now we can compare different queries and discover how similar they are. Further on, by discovering similarities, we can generate some recommendations to the user or exploit those information for system optimization and similar. We call ***bridging*** to the process aimed at identifying how similar two NMACs are. By adding some multidimensional operators to one NMAC we can obtain the other one and thus *bridge* it one with another. Based on the length of the bridge found, we can decide whether it makes sense or not to bridge the two queries. Other usages of normalization and bridging will be discussed later in appropriate sections.

3.2 Framework

In this section it is presented in detail the framework of using the Multidimensional Algebraic Structure of analytical queries. This approach is supposed to present efficient and powerful way to store information about queries so that that information can later be exploited. The phases of the framework are Obtaining the MAC from a SQL query, Normalizing MAC and last one, Bridging which is presented as a concept for further analysis and usage in future.

3.2.1 Obtaining the MAC from a SQL query

Professors Oscar Romero and Alberto Abello have shown in [9] how multidimensional operators can be expressed in terms of restricted operators of the relational algebra. We take advantage of this work to identify the MDA operators, given an SQL query. First, it is briefly refreshed the relationship between both algebras and later, we discuss how to formulate the MAC of an SQL query. Without loss of generality, we denote by raw data (over which the MDA operators are applied) to the universal relationship of the tables in the FROM clause of the query. The universal relation contains all fields defined in all source tables.

Table 4. Comparison table between the relational and MD algebras

Reference Operator		"Selection"	"Projection"	"Join"	"Union"	"Group by"	"Aggregaton"
Selection		χ_{Descs}					
Projection			$\chi_{Measures}$				
Roll-up						$\chi_{Descs_{id}}$	$\chi_{Measures}$
Drill-across			$\chi_{Descs_{id}}$	$\chi_{Descs_{id}}$			
changeBase	**Add Dim**			$\chi_{Descs_{id}}$			
	Remove Dim.		$\chi_{Descs_{id}}$				
	Alter Dim.		$\chi_{Descs_{id}}$	$\chi_{Descs_{id}}$			
Union					χ		

Table 4 summarizes the mapping between both sets of algebraic operators. Columns represent relational algebra operators while row represent MDA operators. Note that we are

considering the extended operators of the relational algebra as in [12]. We use the following notation in the table:

- $x_{Measures}$ - if the MD operator is equivalent to the relational one but it can be only applied over measures
- x_{Descs} - if the MD operator must be applied over descriptors
- $x_{Descs_{id}}$ - if MD operator can be only applied over level identifiers
- x - MD operator is equivalent with relational operator without additional restrictions

If the translation of a MD operator combines more than one relational operator, both appear ticked in the same row.

It is important to state also the constraints of the MD model that affect the usage of these operations:

1. Fact/Dimension dichotomy must be preserved, which is reflected in that descriptors and measures are disjoint.
2. Summarizability necessary conditions (as in [13]) must be preserved, which is reflected in the multiplicities of relationships used in the operations as follows:
    a. **Roll-up**: $level_i \rightarrow level_j$ must be one-to-many (or many-to-one, if it actually corresponds to a drill-down operation).
    b. ChangeBase: $base_1 \rightarrow base_2$ must be one-to-one.
    c. Drill-across: $cube_1 \leftrightharpoons cube_2$ must be one-to-one.

The first item can be easily validated, whereas testing the cardinalities in the second one is reduced to discover *functional dependencies* among the set of attributes involved in the relationship, by sampling the relational source. Note that we are able to formulate the MAC by directly applying this result (we address the reader to [9] for a detailed justification of this table). If an SQL query cannot be fully formulated in terms of MDA operators it means that, according to MDA, it does not make MD sense and thus, it should be discarded.

Now, we present the example of extracting MDA operators for MD schema in Figure 6 and SQL query:

```
SELECT Date.year, Location.country, AVG (Sales.price)
FROM Sales, Date, Location
WHERE Sales.product = '1' AND Sales.day = Date.day   AND Sales.store =  Location.store
GROUP BY Date.year, Location.country;
```

We first have *raw data* consisting of tables Sales, Date and Location, as the universal relation. Next we identify MDA operators and present how their appliance manifests to the output data cube schema. Note that in output data cube schema set of dimensions we denote with [D], set of measures with [M] and set of slicers with [S]. The identified MDA operators are:

- Selection over Product dimension – more formally, $\sigma_{Sales.product='1'}$, identified by "Sales.product = '1'" from the WHERE clause of the SQL query.
  - Manifest to the output data cube schema: {Sales.product='1'} [S]
- ChangeBase from the key of universal relation of Date, Product and Location dimension tables to Location and Date dimension – more formally,

  $\chi_{Date.day,Product.product,Location.store \to Date.day,Location.store}$, identified by presents of only Date and Location dimensions in the SELECT clause of the SQL query and universal schema of the data cube.
  - Manifest to the output data cube schema: {Date.day, Location.store}[D], {Sales.product='1'}[S]
- Projection of Sales.price measure – more formally, $\pi_{Sales.price}$, identified by "AVG(Sales.price)" from the SELECT clause of the SQL query.
  - Manifest to the output data cube schema: {Data.day, Location.store}[D], {Sales.price}[M], {Sales.product='1'}[S]
- Roll-up from Date.day to Date.year – more formally, $\gamma^{Date.day \to Date.year}_{avg(Sales.price)}$, identified by "GROUP BY Date.year" part and "AVG(Sales.price)" from the SELECT part of the SQL query.
  - Manifest to the output data cube schema: {Data.year, Location.store}[D], {AVG(Sales.store)}[M], {Sales.product = '1'}[S]
- Roll-up from Location.store to Location.country – more formally, $\gamma^{Location.store \to Location.country}_{avg(Sales.price)}$, identified by "GROUP BY Location.country" part and "AVG(Sales.price)" from the SELECT part of the SQL query.
  - Manifest to the output data cube schema: {Data.year, Location.country}[D], {AVG(Sales.store)}[M], {Sales.product = '1'}[S]

The MAC of this SQL query would be:

- $\gamma^{Location.store \rightarrow Location.country}_{avg(Sales.price)}(\gamma^{Date.day \rightarrow Date.year}_{avg(Sales.price)}(\pi_{Sales.price}(\sigma_{Sales.product='1'}($

  $\chi_{Date.day,Product.product,Location.store \rightarrow Date.day,Location.store}(raw\_data)))))$

An example, of binary MDA operator, in this case union, we can show for next query:

SELECT Date.year, Location.country, AVG (Sales.price)
FROM Sales, Date, Location
WHERE Sales.product = '1' AND Sales.day = Date.day AND Sales.store = Location.store
GROUP BY Date.year, Location.country
UNION
SELECT Date.year, Location.country, AVG (Sales.price)
FROM Sales, Date, Location
WHERE Sales.product = '2' AND Sales.day = Date.day AND Sales.store = Location.store
GROUP BY Date.year, Location.country;

The MAC of this SQL query would be:

- $\gamma^{Location.store \rightarrow Location.country}_{avg(Sales.price)}(\gamma^{Date.day \rightarrow Date.year}_{avg(Sales.price)}(\pi_{Sales.price}(\sigma_{Sales.product='1'}($

  $\chi_{Date.day,Product.product,Location.store \rightarrow Date.day,Location.store}(raw\_data)))))$

  $\cup$

  $\gamma^{Location.store \rightarrow Location.country}_{avg(Sales.price)}(\gamma^{Date.day \rightarrow Date.year}_{avg(Sales.price)}(\pi_{Sales.price}(\sigma_{Sales.product='2'}($

  $\chi_{Date.day,Product.product,Location.store \rightarrow Date.day,Location.store}(raw\_data)))))$

A MAC represents the MD counterpart of the analytical SQL statement analyzed. Starting from the raw data, by applying MD operators over that raw data we modify data cube to obtain final cube schema as a result of the query.

By definition, a MAC is a tree-shaped structure. Like in the relational algebra, this is because of binary operators. The grammar capturing its semantics is as follows (χ, σ, π, γ, $\cup$, $\bowtie$ represent the MDA operators; see Section 3.1.1):

$MAC \rightarrow$ rawData NP | $(MAC \cup MAC)$ NP | $(MAC \bowtie MAC)$ NP $\quad Q \rightarrow CB\ S\ R\ P$

$NP \rightarrow Q \mid Q\ NP$ $\quad CB \rightarrow 0 \mid \chi CB$ $\quad S \rightarrow 0 \mid \sigma S$ $\quad R \rightarrow 0 \mid \gamma R$ $\quad P \rightarrow 0 \mid \pi P$

From now on, we will talk about the root-side and the leaf-side of the MAC. The tree leafs are raw data (i.e. with no transformations). Furthermore, we call a *navigation path* (NP from here on) to any *partially ordered set of unary operations* consecutive within the tree. These NPs can be thought as data manipulation to produce the desired presentation or alignment (i.e. the data cube MD space –*changeBases*-, slicers –*selections*-, data granularity produced –*roll-ups*- and subset of measures shown –*projections*-), whereas nodes collapsing two *branches* (from here on, we simply refer to the input NPs of binary operators as branches) are generating a new set of tuples (if desired, we may keep manipulating the result with a new NP). Thus, note that a single MAC can contain more than one NP.

Indeed, data might need to be aligned before being able to collapse them. For example, we may need to roll-up to the same granularity level before uniting or drilling-across data from two different cubes (i.e. align the input branches of binary operators to produce the one-to-one relationship demanded by *union* and *drill-across*).

In previous example we had two NPs as input branches of union operator.

Finally, we talk about the *pivotal node* as the tree node dividing the MAC into two well-differentiated layers: the *structural layer* and the *presentation layer* (i.e. the first binary ancestor of the root). In other words, the pivotal node identifies the set of tuples (i.e. the *structural* part) over which we only apply unary operations (i.e. a NP representing how data is *presented* to the user). In previous example, the pivotal node is the union, because in this case it represents the structural part. We do not have presentation layer in this case, since it is also the root.

3.2.2 Normalizing MAC

Once we have formulated the MAC for a given statement, we aim at normalizing it. In our approach we benefit form the algebraic structure proposed, and we use a set of equivalence rules to pull the MD operators up the algebraic structure. Thus, the MDA equivalence rules (shown in Table 5, Table 6 and Table 7) are an immediate consequence of considering the MDA operator semantics over the relational algebra equivalence rules (explained in [12]) and considering the constraints introduced in Section 3.2.1.

Table 5. MDA equivalence rules for normalizing NP

Operator	Projection	Roll-up	Selection	ChangeBase
Projection	×	×	×	×
Roll-up	✓	~	×	×
Selection	✓	✓	✓	×
ChangeBase	✓	~	~	✓

Table 6. MDA equivalence rules for removing MD operators from NP

Operator	Projection	Roll-up	Selection	ChangeBase
Projection	✗	✓	✓	✓
Roll-up	✗	∼	∼	∼
Selection	✗	✗	✓	∼
ChangeBase	✗	✗	✗	✓

Table 7. MDA equivalence rules for binary operators

Operator	Projection	Roll-up	Selection	ChangeBase
Drill-Across	✓	↗↖	↗↖	↗↖
Union	↗↖	↗↖	↗↖	↗↖

MDA equivalence rules are split into three tables for the purpose of clarity in case of tables 5 and 6, and also because table 7 has different interpretation than tables 5 and 6 as it is explained later.

Tables 5 and 6 reflect the rules that apply to linear structures (NPs) of the MAC and as such they indicate if column operator *can be pulled up* (exchange place with) the operator in the row so it would be closer to the MAC's root (MAC is "vertical structure" – a tree, so for that reason we use term "pull up"). The meaning of each cell in tables 5 and 6 is the following:

- "✓" – the MDA operator in the column *can be pulled up* the operator in the row
- "✗" – there is a conflict, so the MDA operator in the column *cannot be pulled up* the operator in the row
- "∼" – there is a partial conflict such that column operator can be pulled up whenever the row operator does not remove the attribute needed by the column operator. In tables 5 and 6 we have next particular cases of pulling column operators up row operators:
    - Roll-up over Roll-up – if two Roll-up operators belong to the same tables we cannot exchange their places because their order must be preserved
    - Roll-up over ChangeBase – if ChangeBase operator changes the dimension over which the Roll-up is done we cannot exchange their places
    - Selection over Roll-up – in this case, we can have conflicts only if we allow Selection operators to be done over non-identifier dimension descriptors. If

we allow that, conflict appears when Roll-up operator removes descriptor that is used by Selection operator and we cannot exchange their places.

- Selection over ChangeBase – if the ChangeBase operator removes dimension descriptor which is used by Selection operator we cannot exchange their places.
- ChangeBase over Roll-up – if the dimension used by Roll-up operator is not changed by ChangeBase operator (that means that it is present in both bases of ChangeBase operator) then we can exchange their places, otherwise we cannot.
- ChangeBase over Selection – if the Selection operator uses the dimension descriptor that is not changed by ChangeBase operator (that means that it is present in both bases of ChangeBase operator) then we can exchange their places, otherwise we cannot.

Table 7 reflects the rules that apply to the binary node of the MAC (which is a tree structure as we said). These rules indicate if column operator *can be pulled up* from branch to the leaf-side of the parent NP of the binary operator. The meaning of each cell in table 7 is the following:

- "✓" – the column MDA operator in the branch *can be pulled up* to the parental node, over the binary operator in the row
- "↗↖" – the same column MDA operator must appear in both branches so it can be pulled up from the branches to the parental binary node (over the binary operator).

Note that we assume well-formedness of MAC in the sense that no attribute is used in an operation if it is not present in the output schema of the previous operation(s).

The final aim of normalization is to distinguish between operators producing the set of tuples retrieved by the query (i.e. the structural layer) and operators manipulating these tuples before being presented to the user (i.e. the presentation layer). However, as discussed in Section 3.2.1, MACs can contain more than one NP (some of them interleaved in the structural layer), although only the root-most NP (i.e. the one amid the pivotal and root nodes) represents the presentation layer. Thus, we *normalize* MACs by *pulling operators in the NPs of the structural part to the presentation layer* (i.e. to the MAC root-side), and we do so by applying Table 5, Table 6 and Table 7.

Interestingly, note that, unlike the relational algebra logical optimization that aims at pushing operators as much as possible in the direction of leafs, we aim at pulling MDA unary operators towards the root. Moreover, we find more ticks in Tables 5, 6 and 7 than we would find

if using the relational equivalence rules and, when something needs to be checked, it is much easier, because in MDA we introduce additional constrains that simplify these rules. For example, we know that the relational selection can be pulled up a projection if the attribute involved in the selection is not projected out by the projection. Furthermore, we know that the MDA projection can only be applied to measures, whereas selection only makes sense over descriptors. Consequently, the MDA projection and selection can always be swapped in a MAC, as the set of attributes involved in each operator will always be disjoint (see Section 3.2.1 for further details). In the general case, special difficulties arise dealing with the relational group by (basis of roll-up, OLAP key operator). Interestingly, we want to remark the gain when dealing with our restricted group by (i.e., roll-up) instead of the generic one, whose difficulty is recognized in [12] (where it is explicitly said that no law is stated) and especially in [14] (where the whole work is devoted to analyze all possibilities only between join and group-by).

The normalization algorithm is just a postorder traversal of the *MAC*, considering that the nodes to visit are *NPs* and binary operations. We then deal with these two kinds of nodes in a different way:

a) For each NP we visit, for each unary operator it contains (from root-side to leaf-side), we pull it up in the direction of the root as much as possible, following the rules in the Table 5.

b) For each binary operation we visit, according to Table 7 if both left and right branches are non-empty *NPs* and some operation coincides in their topmost *Q* which can be pulled up through its successors in *Q* according to the Table 6., that unary operation is removed from both branches and added at the leafs-side of the *NP* in the binary parent node.

As a result of this algorithm, we say that a NMAC is a MAC with the following properties:

i) Many *NP* can appear in a *MAC*. *NPs* stuck in the structural part are needed for aligning the inputs of binary operators (and not for presentation purposes).

ii) Many *Q* may appear at each *NP*, but the minimum number would be generated, each potentially containing χ, σ and γ in this order.

iii) π can only appear in the topmost *Q* of every *NP*, and following the order imposed by the containment of attributes.

For MAC from previous example:

- $\gamma^{Location.store \rightarrow Location.country}_{avg(Sales.price)}(\gamma^{Date.day \rightarrow Date.year}_{avg(Sales.price)}(\pi_{Sales.price}(\sigma_{Sales.product='1'}($

  $\chi_{Date.day,\Pr oduct.product,Location.store \rightarrow Date.day,Location.store}(raw\_data)))))$

  $\cup$

  $\gamma^{Location.store \rightarrow Location.country}_{avg(Sales.price)}(\gamma^{Date.day \rightarrow Date.year}_{avg(Sales.price)}(\pi_{Sales.price}(\sigma_{Sales.product='2'}($

  $\chi_{Date.day,\Pr oduct.product,Location.store \rightarrow Date.day,Location.store}(raw\_data)))))$

Normalization algorithm first normalizes corresponding MACs of the branches and we obtain:

- $\pi_{Sales.price}(\gamma^{Location.store \rightarrow Location.country}_{avg(Sales.price)}(\gamma^{Date.day \rightarrow Date.year}_{avg(Sales.price)}(\sigma_{Sales.product='1'}($

  $\chi_{Date.day,\Pr oduct.product,Location.store \rightarrow Date.day,Location.store}(raw\_data)))))$

- $\pi_{Sales.price}(\gamma^{Location.store \rightarrow Location.country}_{avg(Sales.price)}(\gamma^{Date.day \rightarrow Date.year}_{avg(Sales.price)}(\sigma_{Sales.product='2'}($

  $\chi_{Date.day,\Pr oduct.product,Location.store \rightarrow Date.day,Location.store}(raw\_data)))))$

And after normalization of branches, we normalize the parent and obtain NMAC:

i) $\pi_{Sales.price}(\gamma^{Location.store \rightarrow Location.country}_{avg(Sales.price)}(\gamma^{Date.day \rightarrow Date.year}_{avg(Sales.price)}(\sigma_{Sales.product='1'}($

$\chi_{Date.day,\Pr oduct.product,Location.store \rightarrow Date.day,Location.store}(raw\_data))$

$\cup \sigma_{Sales.product='2'}($

$\chi_{Date.day,\Pr oduct.product,Location.store \rightarrow Date.day,Location.store}(raw\_data)))))$

Finally, we should normalize the presentation layer, but it already is.

3.2.3 Bridging NMACs

Working with algebraic expressions under normal form makes it easier to detect if, syntactically, two expressions are similar to each other. In our context, similar NMACs may be considered logically related from an analytical point of view, and if two NMACs are *close enough* to each other, they are considered to belong to the same analytical session. In that case, they are *coalesced* into a session and both NMACs are logically related by *annotating* their *bridging operators*. Formally, given two NMACs n_1, n_2, we say we can bridge them if by means

of some MDA operators (the bridging operators), we can transform the output of n_1 into that of n_2.

In our current approach we only analyze those queries whose *structural part* coincides by comparing their *presentation layers* (both concepts have been previously introduced in Section 3.2.1). Let P_1 and P_2 be the presentation layer of n_1 and n_2, respectively, and CS_1 and CS_2 their data *cube schemas*. A cube schema (introduced in Section 3.1) is a MD interpretation of the output produced by each query. According to the MDA semantics, we can characterize it as follows (see Section 3.2.1): (i) the set of measures (i.e. data) shown to the user; (ii) the set of dimensional attributes selected to produce the MD space at a certain granularity level and (iii) the set of slicers applied. Now, we take advantage of the MDA minimality (operators cannot be derived by composition) and closeness (their output is a cube) properties. Since MDA is close and every operator has its inverse, by definition, we can transform CS_1 into CS_2 by means of a finite set of MDA operators (in the worst case, it would entail to *undo* all the operators that lead to CS_1 and *redo* those in CS_2). Furthermore, given its minimal property, we know which operators can be applied in order to align each cube schema feature. In other words, we can split the comparison of P_1 and P_2 into smaller comparisons regarding the cube schema part affected by the MDA operators:

i) **Measures:** Let $m1_1,...,m1_n$ and $m2_1,...,m2_t$ the list of measures in CS_1 and CS_2, respectively.
   - If $m1_1,...,m1_n$ and $m2_1,...,m2_t$ coincide nothing has to be done.
   - $\forall m1_i \in CS_1$, s.t. $m1_i \notin CS_2$ the corresponding projection disregarding $m1_i$ is annotated in the bridge between n_1 and n_2.
   - $\forall m2_i \in CS_2$, s.t. $m2_i \notin CS_1$ the corresponding drill-across is annotated to add $m2_i$ is to the output schema.

ii) **MD space:** First, we analyze the relationships between the MD spaces in CS_1 and CS_2.
   - If CS_1 and CS_2 are exactly the same, nothing has to be done.
   - Else, for each one-to-many or many-to-one relationship identified between CS_1 and CS_2 we need to modify the output granularity accordingly. If a one-to-many relationship is identified, a proper drill-down operator is annotated in the bridge. Else, in case of a many-to-one relationship, the corresponding roll-up is added.
   - In any other case, given that the structural part of n_1 and n_2 coincide, a 1-1 relationship, as a whole, should be identified between CS_1 and CS_2. Thus, we

need to navigate from the MD space in CS_1 to the alternative space in CS_2 and the corresponding changeBase is added to the bridge.

iii) **Slicers:** Being p_1 and p_2 the conjunction of the predicates in the selections of n_1 and n_2, respectively.

- If $p_1 \equiv p_2$ nothing has to be done.
- Else if $p_1 \sqsubset p_2$, the proper union(s) is (are) added to the bridge.
- Else if $p_1 \sqsupset p_2$, a selection(s) is (are) added.
- Else a union(s) (to undo p_1) and a selection(s) (to carry out p_2) are added.

Again, note that this algorithm is sound thanks to the MDA properties, which allow us to undo and redo complementary operators (i.e. projection Vs. drill-across, union Vs. selection, roll-up Vs. drill-down and changeBase Vs. changeBase) to produce CS_1 from CS_2. The produced bridge is then evaluated to decide whether n1 and n2 are similar enough, if so we consider both NMACs to belong to the same session. It is out of our current objectives to provide an empirical function to identify when two NMACs are similar enough as to be coalesced in the same session, as it is an application-dependent task. As result, both NMACs are stored in an ordered structure (i.e. a list of NMACs) representing the session and we annotate their relationship with the bridging operators NP_b (to keep track of their logical connection). Finally, we use the last NMAC to keep looking for other queries in the session, whereas the annotated bridging is kept in order to exploit it in future tasks such as query recommendation.

Example of the bridging in case we have to NMACs for queries:

- Q_1:

$$\pi_{Sales.price}(\gamma^{Location.store \rightarrow Location.country}_{avg(Sales.price)}(\gamma^{Date.day \rightarrow Date.year}_{avg(Sales.price)}(\sigma_{Sales.product='1'}(\chi_{Date.day,\Pr oduct.product,Location.store \rightarrow Date.day,Location.store}(raw\_data)))))$$

- Q_2:

$$\pi_{Sales.price}(\gamma^{Location.store \rightarrow Location.country}_{avg(Sales.price)}(\gamma^{Date.day \rightarrow Date.month}_{avg(Sales.price)}(\sigma_{Sales.product='1'}(\chi_{Date.day,\Pr oduct.product,Location.store \rightarrow Date.day,Location.store}(raw\_data)))))$$

The query Q_1 can be bridged with query Q_2, and their cube schemas are exactly the same except for their MD spaces, among which we can identify a many-to-one relationship (form year to month). Thus, a drill-down is annotated as the bridge from Q_1 to Q_2. In this case, it is clear that they are close enough and thus, both NMACs are stored in the same session *s*. Semantically, the annotated bridge means that Q_2's output can be obtained by bridging Q_1's NMAC with the

annotated drill-down (this is represented in the MAC below, where the drill-down is represented by the left-most operator):

- $\gamma^{Location.year \rightarrow Location.month}_{avg(Sales.price)}(\pi_{Sales.price}($
  $\gamma^{Location.store \rightarrow Location.country}_{avg(Sales.price)}(\gamma^{Date.day \rightarrow Date.year}_{avg(Sales.price)}(\sigma_{Sales.product='1'}($
  $\chi_{Date.day,\text{Product.product},Location.store \rightarrow Date.day,Location.store}(raw\_data))))))$

3.3 Usage for query recommendations

After we have seen all the steps of the framework, now we present the intuition how this characterization can be exploited in discovering and characterizing analytical sessions (as we shortly mentioned in the Section 3.2.3 when talking about bridging) and further use analytical session for generating recommendation for the user.

Up to now, as we saw in the Chapter 2, current methods focus on isolated queries, which are analyzed on their own without considering the logical connection analytical queries form the same session do have. However, it is well-known that analytical sessions are formed of related queries capturing the reasoning flow during the analytical session [15]. Accordingly, we propose to *represent* those queries logically connected by means of a single structure capturing the whole session. Below we discuss how we can use this for generating recommendations.

Consider the collaborative recommendation (explained in Chapter 2). Suggestions to the current user can be based on navigations similar to hers. Suppose we can characterize the past sessions over a data cube, as well as the current session, by their NMACs. Using bridging, we can identify the former navigations that are similar to the current one, and the OLAP operations in these navigation paths can be the basis to recommendations for the current navigation.

The idea introduced above can be achieved as follows. Consider the SQL log of the past sessions and the current session. For each past session, all SQL queries are translated into their MDA counterparts.

The current session is bridged with all of NMACs obtained from the log. For each past NMACs, the bridge, being itself an MDA expression, indicates the proximity between the current session and the past sessions, for instance in terms of the number of OLAP operations. For those past NMACs that are the closest to the current session, consider the algebraic operations that compose the bridge. These operations can be used directly to suggest, as a recommendation, these

operations applied to the current query. Alternatively, among these operations, only those that do not change the current query *substantially* can be used.

If we go back to the example in Section 3.2.3, about bridging, and suppose that the last query of the current session is Q_1 and that a past session corresponds to Q_2. Then the drill-down operation corresponding to the bridge between Q_1 and Q_2 can be recommended to the current user.

Techniques for recommending queries to database users or data warehouse users have already been proposed, but to the best of our knowledge, none of them leverage the algebraic structure of the queries to direct the user toward relevant part of the database based on the current navigation. Using navigation path as a basis for enhancing analysis opens interesting perspectives. For example, in addition to the approach outlined above, one interesting research direction is to extract navigational patterns from a user's query log, to learn how the use prefers to navigate (i.e. what are her favorite OLAP operations), and then to suggest queries based on these preferences.

4 Prototyping

This chapter presents the development process of previously introduced framework. Bridging phase of the framework is left on the theoretical basics and considerations regarding to the scope of this thesis. Note that, as mentioned in Section 3.2.3, an empirical function to identify when two NMACs are similar enough needed for the bridging phase is out of the objectives of this framework by now, as it requires further research before implementation. Research and development processes have surprisingly showed that the steps for obtaining MACs and normalizing them are actually crucial for any further use (like query recommendation). When we extract all the information about the queries and keep them in a structured and normalized way, further exploitations of them seem, if I may say, as a straightforward process if the proper structure have been used to store the queries. Precisely because of this, these phases (obtaining MAC and further on NMAC) happened to be hard and tedious tasks. Many new details that needed to be considered showed up while developing this prototype and they contributed in better elaborating theoretical basics. Development process started with the goal of generating query recommendations for the user but due to the difficulties that arisen, it turned out that these preparation steps are sufficiently voluminous.

The rest of the chapter is organized as follows. First, we introduce the methodology used for the development process. Next, both functional and non-functional requirements are listed. Afterwards we explain the design with regards to requirements, architecture, input parameters and expected output. Further on, we focus on implementation details and explain technology and tools used, important aspects of coding, and finally obstacles encountered with corresponding solutions that were found. Last section of the chapter represents testing phase of prototyping.

4.1 Development method

Development method used for the practical part of the thesis is agile software development. This method was chosen because it seemed to be most appropriate for the settings of this research project. Due to the characteristics of one master thesis student project when requirements are often changing, when student is progressively developing the program with regular consultations with his tutors and when the progress and final outcome can be

unpredictable due to the development of something new that no one has ever done, this methodology fits perfectly because of the flexibility it provides.

4.1.1 Agile software development

Agile software development is a relatively new approach. We can find many sources that talk about it. Since [20] gives a concise and comprehensive overview it is used as a reference in this thesis. Some additional details and explanations regarding the agile modeling can be found in [18]. In this section we present the agile software development method together with the comments (where found appropriate and needed) of how it was or was not applied in this project and why.

Agile software development is a software development method based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. The Agile software development process and values are illustrated in Figure 8 and are more elaborated later.

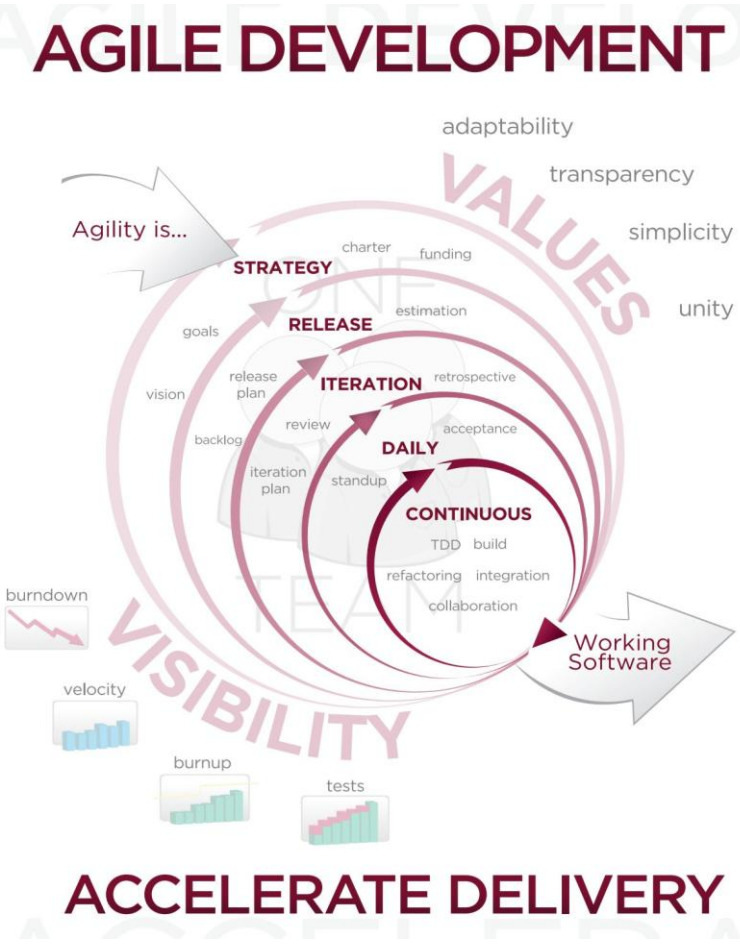

Figure 8. Agile software development poster, taken from [20]

The Agile Manifesto can be found in [19], and it defines the approach now known as agile software development. This method values are:

- **Individuals and interaction** over processes and tools. This state out the value of self-organization and motivation together with cooperation between individuals. Individuals are important factor in this project since there was only one person (a student) directly working on the development of the system. Nevertheless, interaction was equally significant due to the participation of the professors when defining requirements and design concept and especially when facing implementation doubts.
- **Working software** over comprehensive documentation. This state out the value of software that works and has the least bugs over just well prepared documentation. By focusing on seeing first implementation results and then corresponding documentation, this value was also important for this project development.
- **Customer collaboration** over contract negotiation. This state out the importance of stakeholder's involvement during whole process of development due to possible changes in requirements and obscurities. Since the professors can be considered as a stakeholders and customer counterpart and they were involved during the whole system development we may say that this value was also applied.
- **Responding to change** over following a plan. This state out the importance of ability to accept and conduct changes in development. This value was crucial for this project considering its research character. Many times we dealt with innovations and revisions to which we had to respond adequately.

Principles for following the Agile software development include:

- Early and continuous delivery of valuable software – This principle was followed in the process.
- Always welcoming changes in requirements – Essential principle for all projects of research type including this.
- As shorter possible and frequent, working software delivery intervals – By often meetings where we checked the new functionalities added we may say that this was also one of the principles of the development.
- Daily cooperation of coworkers – This was not applied since there was no development team but only one person.
- Trust in motivated individuals – This was one more of the crucial principles for this project since the whole development was entrusted to only one student.

- Face-to-face team conversation – There was no development team but face-to-face communication was preferred for all consultations and discussions.
- Progress measured by working software – Due to a research character of the thesis, which this development project is a part of, the focus was not only on the implementation. Still when working on development, progress was measured by working functionalities.
- Constant and sustainable development – Process of development started from the beginning of the work on this thesis and lasted until the end in parallel with other needed activities.
- Continuous attention to technical excellence and good design – The design of the system was refined several times when some sufficient parts were discarded while other needed were added.
- Essential simplicity – There was a tendency of modularizing the system in important parts as much as possible so that its design can be easily understood and, if and when needed, modified.
- Self-organizing teams – Again, there was no team, but there was flexibility in organizing time as long as the tasks are done.
- Effective use of time – When needed, the deadlines were introduced so the time would be used effectively.

Characteristics promoted by most of the agile development methods are agile development, teamwork, collaboration, and process adaptability throughout the life-cycle of the project. Teamwork, as previously stated, was not applicable in this project since there was no team but other methods were guidelines for the development process.

Agile methods break tasks into smaller incremental tasks that require minimal planning and are done in short time iterations. An iteration goes through a full software development cycle consisting of planning, requirements analysis, design, coding, unit testing and acceptance testing when a working product is demonstrated to stakeholders. This way development process minimizes the risks and allows the project to adapt to changes quickly. Goal of an iteration is to have available release with enough new functionality but it does not have to be the case always. We developed system in iterations and if we consider professors as stakeholders we may say that we completely followed this method. Benefit of this was having system releases as soon as possible, since it is not always predictable how each iteration would evolve.

Teamwork is of high importance and is typically present in form of small size (less than 10 members) teams that are usually cross-functional (each member has responsibility for

achieving some functionality) and self-organized. Preferred and emphasized form of communication is face-to-face. This was not applicable in the case of this project.

Development process also involves a customer representative. This person appointed by stakeholder is in charge of answering any mid-iteration problem-domain questions raised up by developers. At the end of each iteration, stakeholders and the customer representative review progress and re-evaluate priorities with a view to optimizing the return on investment and ensuring alignment with customer need and company goals. Again preferred communication is face-to-face. In our case, professors can be considered as both stakeholders and customer representatives.

This development method emphasizes working software as the primary measures of progress as we can find in the above mentioned principles. This principle together with the preference for face-to-face communication produces less written documentation than other methods. Agile methods focus on adapting quickly to changing realities.

Almost all agile methods are suitable for method tailoring – adapting method fragments determine a system development approach for a specific project situation. The practical implication of this is that agile methods allow adaptation of working practices according to the needs of individual projects. In the case of project for the practical part of this master thesis, some of these agile software development process main manifests are:

- Changes in requirements were non rare due to the research character of the project so handling them was of crucial importance.
- The project was developed bit by bit, in iterations.
- At meetings with my tutors we discussed results of previous iteration and made plans for next iteration of what needs to be done and how to achieve it.
- These meeting were important for easy face-to-face communication, followed by regular email communication.
- Meetings were frequent and important for constant progress.

4.2 Requirements

General requirement in this project was to develop a system that can efficiently extract and normalize the MAC out of a SQL query. Since this system is not supposed to be user interactive, but modeled as a batch process (as it is explained in next Section 4.3) use case diagrams or visual prototypes are not convenient for gathering and representing requirements.

Use case diagrams, because there is not essential interaction with a user or any other system at this version of our system. Graphic user interface created is just for the presentational purposes. Neither visual prototypes are convenient since we focus on analyzing and storing SQL queries which is a user independent task in a sense that we do not need user's input how to analyze and store the queries. Further on, requirements for this system are not characterized by being numerous but by being complex (not all of them of course). Since the requirement do not go into implementation and how the thing are supposed to be done, but what functionalities and characteristics system should provide we chose to represent them in a form of simple and graspable list. This list of requirements comes from the discussion at the meetings we had and it was revised, updated and adapted to the theoretical findings related to framework during the development of this system. Requirements we further categorize into two well known categories which are functional and non-functional requirements. Functional requirements define specific behavior or functions of the system while non-functional requirements (also know as quality requirements) impose constraints on the design or implementation (such as performance requirements, security, or reliability) [24].

Functional requirements for this system are:

- The system needs to be able to parse SQL queries.
- The SQL queries have to be analytical SQL queries.
- The system needs appropriate data structures for storing information about each individual multidimensional operator.
- The system needs to extract the information about each multidimensional operator out of the parsed SQL query.
- The system needs to store extracted information about each multidimensional operator in appropriate data structure.
- The system needs appropriate data structure for storing information about MAC obtained from a parsed SQL query.
- The system needs to extract the information about MAC from a parsed SQL query by identifying all the multidimensional operators belonging to that SQL query
- The system needs to store extracted information about MAC in appropriate data structure.
- The system needs to provide functionality of normalizing extracted and stored MAC and in such a way obtain the NMAC.
- For the purpose of presentation of system's functionalities, the system needs to provide graphic user interface that will enable:

- choosing the file containing SQL query
- graphic presentation of MAC extracted from chosen SQL query
- graphic presentation of NMAC obtained by normalizing previously obtained MAC

Non-functional requirements for this system are:

- The system needs to have good performance, primarily fast response time due to future exploitation where it should provide real-time, interactive feedback.
- The system needs to be developed in such a way that enables future extension
- The system needs to be well documented for the purposes of future development, upgrade and use.

4.3 Design

Accordingly to the previous stated requirements the design of the system and corresponding data structures that should capture the knowledge obtained are modeled. This system is modeled to be used in batch processing and because of that it does not provide many possibilities for user interaction. According to [21], batch processing is execution of a program or sets of programs on a computer without manual intervention. A program takes a data input file/files, processes the data, and produces a set of output data which may be presented to the user if needed. In our case, a user is supposed to submit a SQL query to some DBMS which returns the result, while our system in parallel processes the query and extracts information about the query that will enable future exploitations. Due to this characteristic of the system there is not many use case diagrams. Instead, we focus on the data structures and corresponding system algorithms that work over these data structures.

The data structures are first presented in less formal way, either by sketch, either by textual description, with the purpose of just grasping the main ideas. Later, appropriate UML diagrams are presented as a design of concrete system. Further on, presentations of significant algorithms and discussions about some important issues are provided.

4.3.1 Data structures

The goal of this section is to present main sketches of the data structures needed in the system. Since this system's primary task is to extract and handle information about a SQL query, data structures in which this information is stored represent an important concepts for later system

design and implementation. During the iterative development of the system there were many minor or major changes and variations of these concepts. Therefore we present final versions of these data structures with some previous versions where appropriate to illustrate the research evolvement. Data structures are presented in form of diagram or textual description.

Before moving on the data structures it is useful to first once again analyze the process in which these data structures are needed. High level abstraction of the modeled process is shown in Figure 9. On this level of details it represents the general idea of having data over which we apply operators that manipulate that data. Therefore, first idea about the multidimensional operators was that information about them could be stored in the data structures of the same type and different internal content that stores data specific for each operator. This internal content corresponds to the operator schema introduced in Section 3.1.1. The data manipulated corresponds to the schema of the data cube introduced in Section 3.1. When over a data cube schema we apply operator schema it produces an output data cube schema which would further on be an input data cube schema over which we apply next operator schema and so on. However due to the existence of the binary multidimensional operators (Union and Drill-Across) this was not possible. These operators need two data cube schemas over which they apply their operator schema. Clearly this high level abstraction of the process could not be directly mapped into similar linear data structure that would capture the knowledge obtained about a query in the process. That is why we had to think of a more appropriate solution.

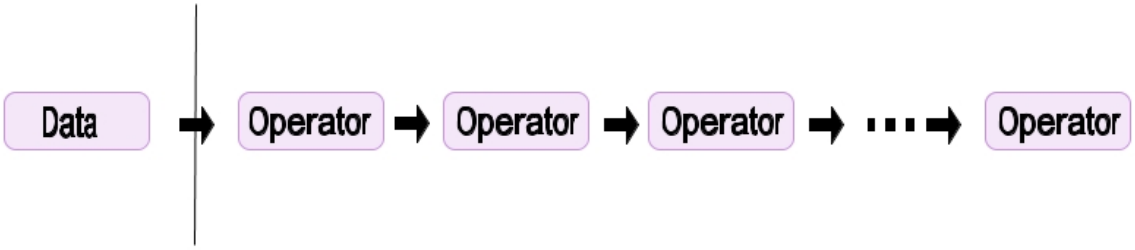

Figure 9. High level abstraction of the modeled process

Obviously, before modeling this data structure that contains all the information about a query, first we need to focus on peculiarities related to the operator and data structures that keep information about them. In that purpose now we introduce the data structures for keeping information about operator schema of the unary operators (check Section 3.1.1 for details) and Raw Data (introduced in Section 3.1.2). From now on, these data structure we will refer to as unary elements. Data structures for storing information about operator schemas of binary operators we introduce later in this section (binary elements from here on). Unary elements with the information about operator schema needed to be stored are:

- Selection element – Information that is stored about this element is:
  - selection predicate such as *attribute = 'value'*.
- Roll-up element – Information that is stored about this element are:
  - the list of the measures (which will always be aggregation functions)
  - identifier of the level from which are we rolling up and
  - identifier of the level to which we are rolling up.
- Projection element – Information that is stored about this element is:
  - the list of projected measures coming form same table which can be either aggregation functions either table attributes.
- ChangeBase element – Information that is stored about this element are:
  - the list of all table attributes belonging to the dimensions from which we are switching and
  - the list of table attributes belonging to the dimensions to which we are switching.
- Raw Data element – Information that is stored about this element is:
  - the list of table names that represent the universal relation of the SQL query.

Detail of how exactly we identified these unary elements in one SQL query are explained later in the Section 4.4.3.

Organization of these unary elements, together with the binary elements in a structure that would adequately represent MAC is next step in the process. Binary tree due to existence of binary elements was a natural choice. Leafs of the tree are the Raw Data unary elements that represent initial data cube schemas. Parental node of a leaf is either unary element either binary element. In case of a unary element, it applies its operator schema over leaf's data cube schema and as an output produces output data cube schema (as presented in example in Section 3.2.1). Similarly, binary element needs two input leafs' data cube schemas over which it applies its operator schema and as output provides result data cube schema. Further on these unary or binary element can have parental nodes that are either unary or binary elements which take their output data cube schema and apply the same procedure. Example of the first version of this data structure for storing information about query's MAC is shown in the Figure 10. This version seems to represent model of MAC in appropriate way.

However, due to differences between binary and unary elements and also to achieve some benefits when normalizing the MAC to obtain NMAC, a modified version of this structure was chosen. Example of this modified version that is also the final version is shown in the Figure 11.

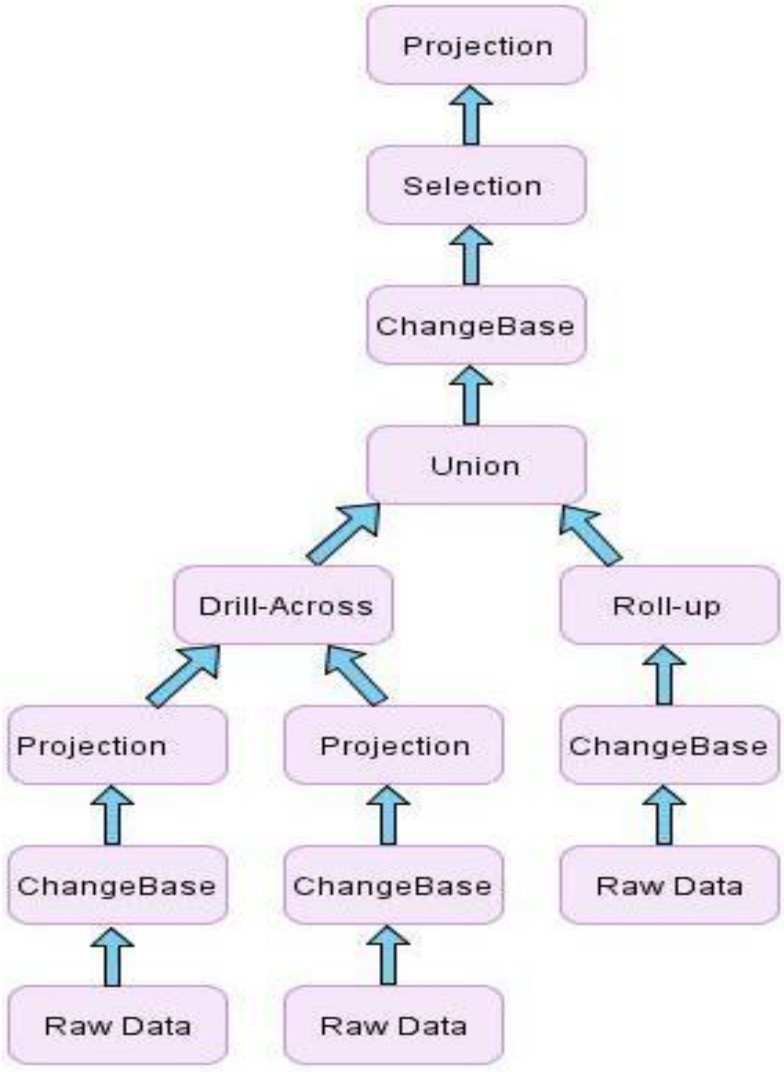

Figure 10. Example of tree data structure for storing MAC, v1

In the final version of tree data structure for storing information about MAC we can find four types of nodes: Raw Data Node, Unary Node, Drill-Across Node and Union Node. These nodes are either container for unary elements, either both binary elements and the containers at the same time, as we will see in short a while. Reasons for having these four types of nodes are different meanings of each node and benefits gained for normalization process. Thus, more precisely:

- Raw Data Node – introduced in the data structure because we always have a Raw Data unary element in each leaf of the tree. Raw Data unary element is never moved from the leaf so in this way we distinguish Raw Data Node from all others which we

consider in normalizing process. Raw Data Node can contain only Raw Data unary element.

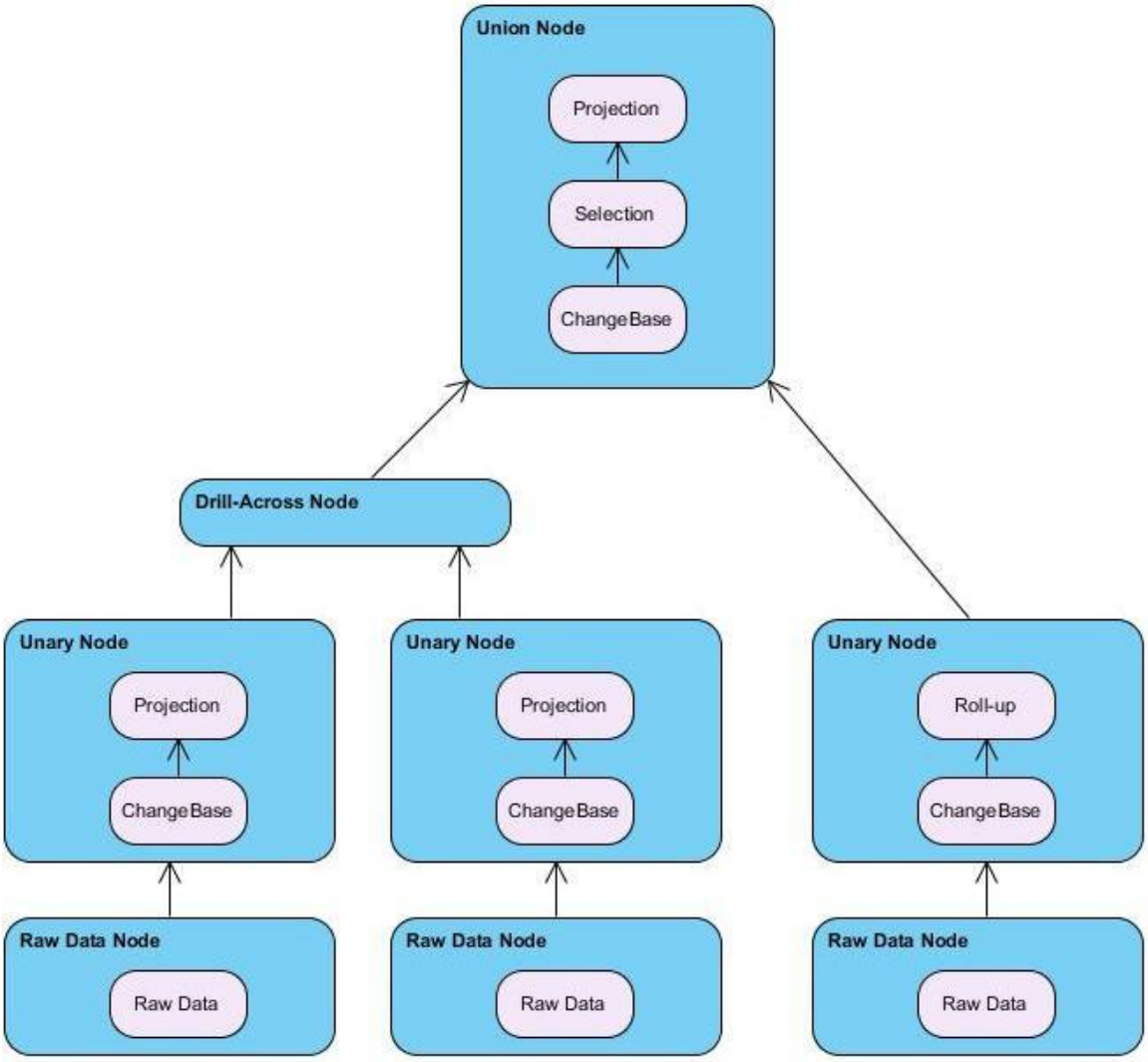

Figure 11. Example of final tree data structure for storing MAC

- Unary Node – introduced in the data structure to model a sequence of unary operators that corresponds to NP concept of the framework. This sequence of operators represents navigation and manipulation over only one data cube. Unary Node can contain all unary elements except Raw Data unary element.
- Drill-Across Node – this node itself represents also Drill-Across binary element. There is no need for additional data structure for storing operator schema of Drill-Across binary operator since the only needed information are two input data cube schemas that are already contained in Drill-Across Node data structure. This Drill-

Across Node may contain inner list of unary elements (without Raw Data unary element) that corresponds to the NP of the framework. Output data cube schema of Drill-Across binary element represented by Drill-Across Node is generated according to operator's semantics presented in the Section 3.1.1 and it will be input data cube schema for the first leaf-side unary element of the inner NP if exists, or for the parental node if exists, or otherwise it is the final output data cube schema of the MAC it belongs to.

- Union Node – this node itself represents also Union binary element. There is no need for additional data structure for storing operator schema of Union binary operator since the only needed information are two input data cube schemas that is already contained in Union Node data structure. This Union Node may contain inner list of unary elements (whitout Raw Data unary element) that corresponds to the NP of the framework. Output data cube schema of Union binary element represented by Union Node is generated according to operator's semantics presented in the Section 3.1.1 and it will be input data cube schema for the first leaf-side unary element of the inner NP if exists, or for the parental node if exists, or otherwise it is the final output data cube schema of the MAC it belongs to.

Benefit that this structure enables for normalization phase is that we can easily normalize it. Normalization is done by normalizing node by node in postorder traversal of the tree. Normalization of one node depends of its type. Before being normalized, MAC will contain only binary nodes that are empty. Later on by normalization process some unary operators might be moved up from children nodes into parental binary nodes. This concept of MAC tree data structure was the one we used in the system design (but with the small difference that in designing we create n-ary tree as it will be explained).

4.3.2 System design

After we have seen general concepts important for organization of system's architecture, now we focus on UML diagrams formally representing that architecture. Level of details on diagrams tends to be just as much needed for understanding of system and its organization while higher details about most important functionalities (class methods) are presented in section about the implementation. System design has changed through the iterations of system development. Some concepts were added, some discarded and in this section we present final design version. Of course, this current final version might be upgraded and changed in future when needed.

Unified Modeling Language (UML), as one of the most-used standards for visualization of application structure and organization, is used in this document as a way to clearly display concept even for readers that are not that much familiar with it. Also, important aspect of UML diagrams is that they enable generalization of object-oriented software engineering ideas so that they do not have to be strictly related to any specific implementation technology.

System design is presented from general overview of packages and then moving on to more specific class diagrams explaining internal organization of important packages. Figure 12 displays general organization of packages. Note that external packages (plug-ins) used are not presented since they are presented later in section about used tools.

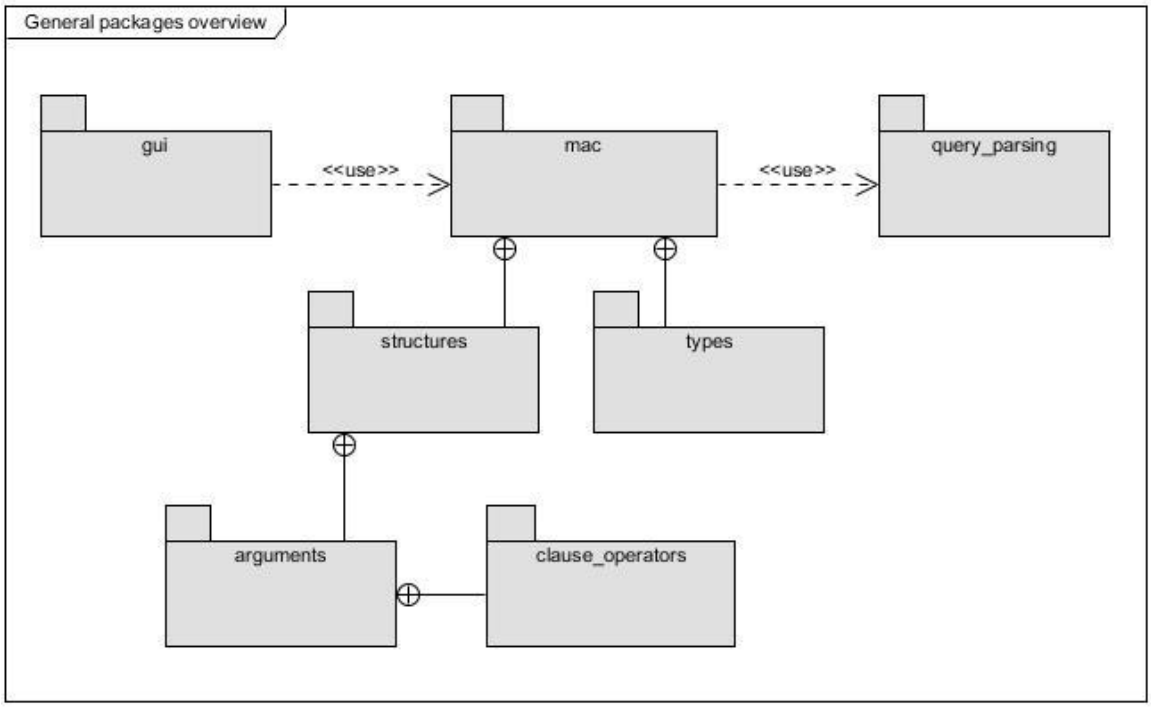

Figure 12. General packages overview

Package *query_parsing* contains classes required for obtaining all the necessary information about the SQL query. SQL query is parsed with the JSQL parser which uses the Visitor Pattern and therefore classes in *query_parsing* package implement required Visitor interfaces needed to extract information about required parts of that query. Since this package depends mainly from the parser used we do not go into details of the classes here but later when talking about the details of parser and Visitor Pattern.

Package *mac* with all contained sub-packages represents the core of the system. This package contains all structure and logic needed for extracting MAC from a SQL query. These

data structures and logic are grouped by sub-packages and in next figures we explain the design of important ones.

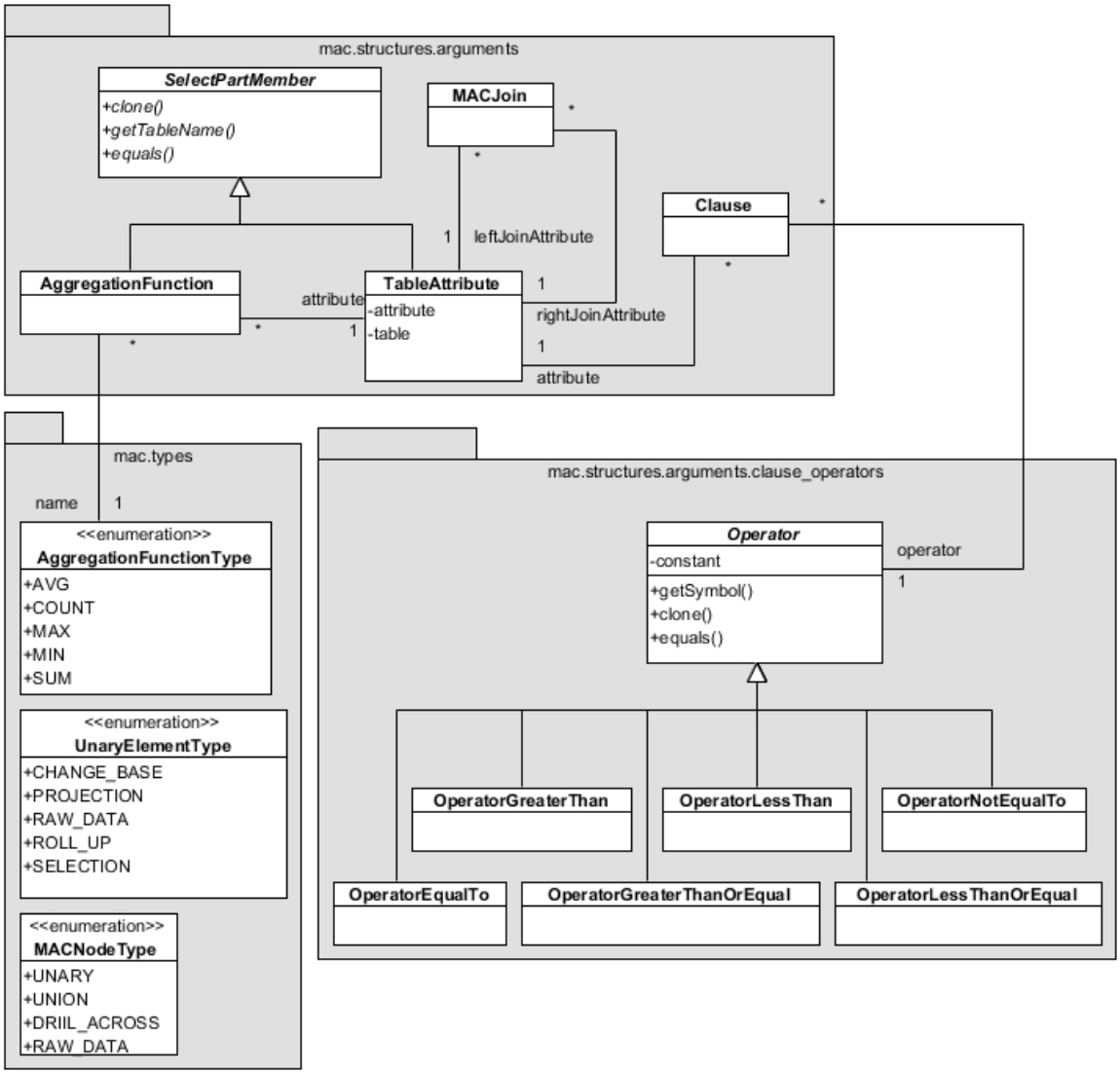

Figure 13. Internal structure of mac inner packages (without mac.structure)

Figure 13 presents the internal structure some of the inner *mac* packages. These packages contain prerequisites needed for *mac.structure* package which contains structures for storing multidimensional operators and MAC. Packages *mac.structures.arguments* and *mac.structures.arguments.clause_operators* represent concepts that store information that we want to extract from a SQL query and at the same time data structures for storing the same.

Package *mac.structures.arguments* contains several classes that model some basic information that we need to store about the SQL query:

- Abstract class **SelectPartMember** represents an abstraction for everything that can be found in SELECT part of SQL query. Abstract operations of this class are *clone()* that returns an identical copy, *getTableName()* that returns table name to which a class is related to and *equals()* which checks if two instances are equal.
- Class **AggregationFunction** that extends **SelectPartMember** represents aggregation functions found in SELECT part of SQL query. This class stores information about type of aggregation function (**AggregationFunctionType**) and table attribute as an argument of the function. It implements all abstract operations of the super class.
- Class **TableAttribute** that extends **SelectPartMember** represents table attributes. This class stores information about table name and attribute name. It implements all abstract operations of the super class.
- Class **MACJoin** represents class for modeling joins found in WHERE part of SQL query. This class stores information about two joined table attributes.
- Class **Clause** represents class for modeling predicates (i.e. clauses) found in WHERE part of SQL query. This class stores information about table attribute and operator (which keeps information about the constant to which we are comparing table attribute).

Package *mac.structures.arguments.clause_operators* contains classes that model operators of our current interest (later, new operators can be added if needed):

- Abstract class **Operator** models operators that we can have in one **Clause**. This class stores information about the constant to which we are comparing some table attribute. Abstract operation *getSymbol()* returns character representation of the Operator, *clone()* returns an identical copy and *equals()* checks if two Operators are equal.
- Class **OperatorEqualTo** extends **Operator** and models "equal to" operator. It implements all abstract operations of the super class.
- Class **OperatorNotEqualTo** extends **Operator** and models "not equal to" operator. It implements all abstract operations of the super class.
- Class **OperatorLessThan** extends **Operator** and models "less than" operator. It implements all abstract operations of the super class.
- Class **OperatorLessThanOrEqual** extends **Operator** and models "less than or equal" operator. It implements all abstract operations of the super class.
- Class **OperatorGreaterThan** extends **Operator** and models "greater than" operator. It implements all abstract operations of the super class.

- Class **OperatorGreaterThanOrEqual** extends **Operator** and models "greater than or equal" operator. It implements all abstract operations of the super class.

Package *mac.types* contains enumerations for various types needed in the system:

- Enumeration **AggregationFunctionType** lists possible types of aggregation function.
- Enumeration **NavigationPathElementType** lists possible types of navigation path elements that we introduce later.
- Enumeration **MACNodeType** lists possible types of MAC nodes that we mentioned already but we later introduce them more formally.

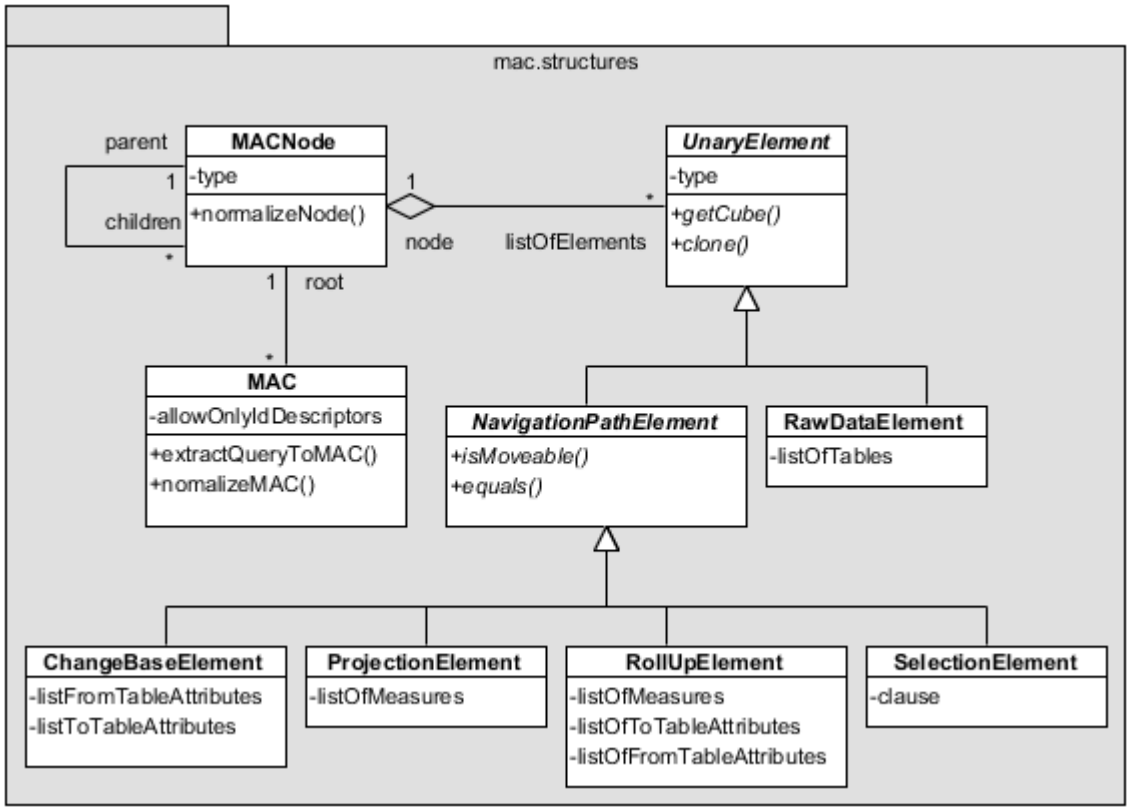

Figure 14. Classes of mac.structures package

Figure 14 shows the internal organization of *mac.structures* package. This is most important package that contains classes for identifying and storing information about multidimensional operators and further on MAC identified. Note that diagram contains only most important concepts (and not all the details). Comments about these package and its elements are:

- Abstract class **UnaryElement** models all the elements that can be found inside the node of the MAC. Attribute type defines the type of the element and is defined by the sub-classes. Abstract operation *clone()* returns an identical copy of the element and abstract operation *getCube()* when implemented by sub-classes it returns the current

data cube (list of dimensions and measures of the data cube). If element is multidimensional operator than it returns data cube after its affect or if it is raw data then it returns the list of tables that represent dimensions of the data cube.

- Class **RawDataElement** models the relation of tables that represent the data cube over which later multidimensional operators (if there are such) are applied. Attribute *listOfTables* is a list of table names mentioned previously.
- Abstract class **NavigationPathElement** is a sub class of **UnaryElement** and super class for all multidimensional operators that can be part of a Navigation Path (for details about Navigation Path check Section 3.2.1). Abstract operation *isMoveable()* checks if a **NavigationPathElement** element can be moved over other **NavigationPathElement** in Navigation Path according to Table 5. Abstract operation *equals()* checks if two **NavigationPathElement**s are equal.
- Class **ChangeBaseElement** is a sub-class of **NavigationPathElement** and is used for storing required (operator schema) information about identified Change Base multidimensional operator. These required information are kept in two lists that are attributes of the class. Attribute *listFromTableAttributes* stores information about all dimensions from which we are changing to new ones stored in other attribute *listToTableAttirubutes.*
- Class **ProjectionElement** is a sub-class of **NavigationPathElement** and is used for storing required (operator schema) information about identified Projection multidimensional operator. Required information is list of Measures (that can be either TableAttribute either AggregationFunction as we explain in later in the section about implementation) and is stored in *listOfMeasures* attribute.
- Class **RollUpElement** is a sub-class of **NavigationPathElement** and is used for storing required (operator schema) information about identified Roll-up multidimensional operator. Required information are three lists – list of measures (as in previous class explained) stored in *listOfMeasures* attribute, list of table attributes that identify dimension from which we are rolling up stored in *listOfFromTableAttributes* attribute and list of table attributes identify dimension to which we are rolling up stored in *listOfToTableAttributes* attribute.
- Class **SelectionElement** is a sub-class of **NavigationPathElement** and is used for storing required (operator schema) information about identified Selection multidimensional operator. Class's attribute *clause* stores information about the predicate that represents the selection over dimensions.

- Class **MACNode** models a tree node according to data structure architecture presented in the Section 4.3.1. In stores next information: attribute *listOfElements* models list of either **NavigationPathElement**s representing NP, either a one element list of **RawDataElement**; attribute *type* stores information about the type of the node (types of the node we can see in Figure 13 as **MACNodeType**); attribute *parent* points at the parent node while attribute *children* which is a list of all nodes that are children nodes. A node can be normalized with operation *normalizeNode()*.
- Class **MAC** is a main class of the whole system and it models a whole MAC tree according to data structure architecture presented in the Section 4.3.1. Class **MAC** in attribute *root* stores information about root node which further on points to the children which point to their children etc. Attribute *allowOnlyIdDescriptors* is a flag that enables or disables selections over id descriptors of the dimensions. Operation *extractQueryToMAC()* builds up a tree out of a SQL query and operation *normalizeMAC()* normalizes obtained MAC.

Package *gui* (Figure 15) is a package created for visual presentation of the MAC obtained from the query. As it was stated earlier, the main goal of this system is not to interact with the user but to analyze user SQL queries, extract MAC and NMAC prepare them for future exploitations so this package has is relatively simple just to present processing that is done. It contains two classes:

- Class **StartScreen** that is first screen created for choosing the file with an SQL query and request extraction of the MAC.
- Class **MACPresentation** that is the visualization for MAC tree created. (jgraph tool was used for this tack)

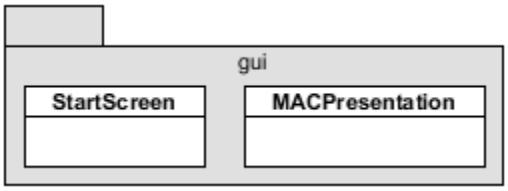

Figure 15. GUI package

4.3.3 Format of the input and expected output

SQL is a declarative computer language and therefore many semantically same queries can be expressed in various syntactical ways. It has a rich syntax that enables different possibilities for manipulating data from the database. However, not all manipulations have sense for multidimensional databases that are of our interest. For a SQL query to have a

multidimensional sense (to retrieve data that can be analyzed from a multidimensional perspective), it has to derive multidimensional schema from the relational sources and possibly apply some multidimensional operators working over that data cube. This brings up the need to define the format of the SQL query that we expect as an input to our system. Note that system does not check if the query makes multidimensional sense since there are already tools that do that but the focus of this system is to identify multidimensional operators and construct MAC as expected output.

The template query (also known as *cube-query*) for the relational database management system that makes multidimensional sense according to [16] is:

SELECT l_1.ID, …, l_n.ID, [F ( ] c.Measure$_1$ [ )], …
FROM Cell c, Level$_1$ l_1, …, Level$_n$ l_n
WHERE c.key$_1$=l_1.ID AND … AND c.key$_n$=l_n.ID [AND l_i.attr Op. *K*]
[GROUP BY l_1.ID, …, l_n.ID]
[ORDER BY l_1.ID, …, l_n.ID]

"Cell" in this SQL query represents set of measures (from the fact table) related to one level for each of its associated dimensions of analysis. The FROM part of template query contains the "Cell table" and the "level tables". These tables are properly linked in the WHERE part of template query. Additionally, the WHERE part of template query can also contain logic clauses restricting a specific level attribute (i.e. a descriptor) to a constant *K* by means of a comparison operator (i.e. equality, inequality, major, minor, etc.). The GROUP BY part of template query shows the identifiers of the levels used to aggregate data. Those columns in the grouping must also be selected in the SELECT part of the template query in order to identify the result (i.e. we must select the multidimensional base to give rise to the multidimensional space). Finally, the ORDER BY clause sorts the output of the query by these identifiers.

In [16] authors introduce this query template as a model for retrieving a Cell (explained at the beginning of previous passage). However, we can take advantage of this query template so that it can be exploited additionally to identify multidimensional operators in the query (check Section 3.1.1 for further details about multidimensional operators). Note that according to [16] this query template covers all possible multidimensional queries.

As an expected output for an input that meets this query template is a MAC and further on NMAC. Our system identifies one by one UnaryElement, creates needed MACNodes and builds up MAC (from those nodes) which can then be normalized. Normalized MAC represents

systematically organized and ordered structure that has a much deeper semantic than just a pure syntactical information about the parts of the query. It keeps information about user's behavior when analyzing query and extracts pure meaning of an SQL query (as already mentioned several SQL queries can have same result, this way we can identify that they are the same). By user's behavior we consider user way of analyzing data cube by changing granularity of data (for example of Figure 6, going from month to year), selecting only certain subsets of data (for example of Figure 6, checking only January values), projecting only some measures (for example of Figure 6, showing only the certain product, only graphic cards in case of an IT store) etc. As we can see, as an output of the system we expect normalized MAC with all the accompanied benefits.

4.4 Implementation

Important complement of any new approach which complements its theoretical part certainly is its practical realization. It supports theoretical concepts, demonstrates the results and validates its functionalities. Implementation can improve and supplement the theory and finally represent the overall result. However, bad implementation can also cast shadow on a good theoretical basis. That is why practical realization of one approach has to be considered as an important part of one potential innovation in the field of work, but it needs to be carefully assessed with all its limitations and properties that can have both positive and negative aspects to the whole project.

Taking into account everything previously said we need to outline some of the basic characteristics of the settings related to this implementation. While features such as technology and tools used we present later in their dedicated sections, here we first note some less technical properties. Once again, an important characteristic of this project is that it is research oriented. Regarding to the implementation it means that system that we are developing is not some ordinary routine where the steps are totally defined but it is just opposite. While implementing the parts of the system we face many new challenges and unpredictability that need to be dealt with. This implies many trial iterations (that are time consuming) needed for reaching final solution presented. Further on, this system is developed by one student with the assistance of his professors. Time period of one semester was just enough to realize the functionalities we did (bear in mind also the time overhead needed for understanding and working on theoretical part of this thesis) but there are still many possibilities for further development and upgrade.

This chapter further on presents used tools and technology, explanations of some important coding elements and finally also very important part of obstacles encountered during implementation together with the corresponding solutions.

4.4.1 Technology

Technology that was used for implementation of the system is Java. Information about Java we can find at many sites and our brief introduction is written according to [22]. Java is a programming language originally developed by James Gosling at Sun Microsystems. It was released in 1995 as a core component of Sun Microsystems' Java platform. Much syntax of the language derives from C and C++ but Java has a simpler object model and fewer low-level facilities. Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture. Java is general-purpose, concurrent, class-based, object-oriented language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere". Java is currently one of the most popular programming languages in use, and is widely used from application software to web applications.

There are five primary goals in the creation of the Java language:

- It should be "simple, object-oriented and familiar".
- It should be "robust and secure".
- It should be "architecture-neutral and portable".
- It should be execute with "high performance".
- It should be "interpreted, threaded and dynamic".

With all these principles java is suppose to provide flexible and powerful solutions for various needs including our system.

4.4.2 Tools

Tools are important factor of every implementation. Efficiency of project development depends quite a lot from their possibilities and their user friendliness. As it is well known many of the software solutions are not free. Because of that for an academic project like this we have to search for available tools that we are allowed to use without any charges. Further on we present the tools that we have used for the implementation.

4.4.2.1 NetBeans IDE

NetBeans IDE is the original Java integrated development environment (IDE). It supports development of all Java application types out of the box. It is a designed as a modular developer tool for a wide range of development tasks. Modules also allow NetBeans to be extended. The base IDE includes an advanced multi-language editor, debugger and profiler integration, file versioning control, and unique developer collaboration features. It has a nice GUI builder which makes easier developing interface for the user. Overall it is stabile and well-supported IDE and as such it was used for building our system.

4.4.2.2 JSQL Parser

JSQL parser is a tool used for parsing SQL statements which are then translated into a hierarchy of Java classes. Then, we copy information needed from JSQL parser's hierarchy of Java classes into our classes presented in the Section 4.3.2. This is a very important part of the system since it is necessary prerequisite. It can influence system's performances. Creating an SQL parser would take a lot of time so it was absolutely necessary to find such a tool. Other SQL parsers that we found were either commercial either not usable and more about that issue we discuss in Section 4.4.4.

This JSQL parser has proven to be efficient enough but it is not very well documented. There is some java documentation of classes and packages but it is poorly explained if explained at all. The most helpful were the examples provided but they also lack of comments. Since there is a plan for future work on this system it would be very useful to explain and make some instructions how this parser works. Nevertheless, I spent many days trying to discover and figure out how this parser works and how can it be used for this systems implementation so in that way it is also the part of the development process.

JSQL parser uses Visitor Pattern for navigation over the generated hierarchy. Therefore we shall first talk about this pattern and then later explain how to use it for obtaining required information.

4.4.2.2.1 Visitor Pattern

In object-oriented programming and software engineering, the **visitor** design pattern is a way of separating an algorithm from an object structure it operates on. A practical result of this separation is the ability to add new operations to existing object structures without modifying those structures. It is one way to easily follow the open/closed principle.

In essence, the visitor allows one to add new virtual functions to a family of classes without modifying the classes themselves; instead, one creates a visitor class that implements all of the appropriate specializations of the virtual function. The visitor takes the instance reference as input, and implements the goal through double dispatch.

While powerful, the visitor pattern is more limited than conventional virtual functions. It is not possible to create visitors for objects without adding a small callback method inside each class. In naive implementations, the callback method in each of the classes is not inheritable.

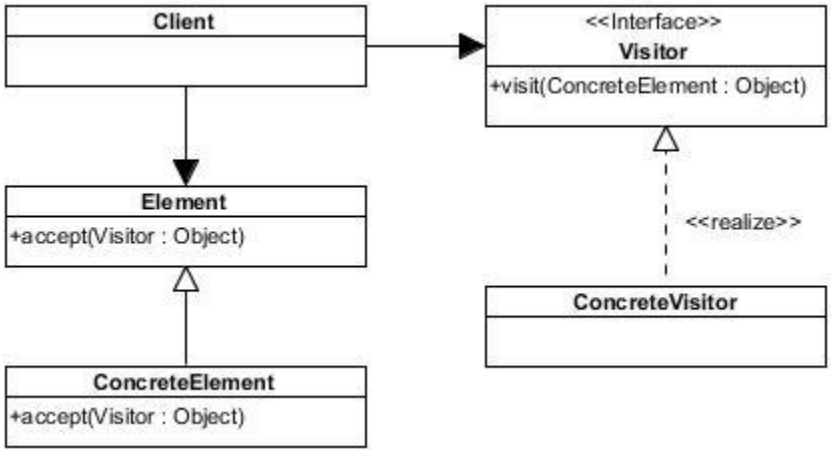

Figure 16. Visitor Pattern

Schema of visitor pattern we can see in Figure 16. A user object receives a pointer to another object which implements an algorithm. The first is designated 'element class' and the latter 'the visitor class'. The idea is to use a structure of element classes, each of which has an *accept()* method taking a visitor object for an argument. visitor is a protocol (interface in Java) having a *visit()* method for each element class. The *accept()* method of an element class calls back the *visit()* method for its class. Separate concrete visitor classes can then be written to perform some particular operations, by implementing these operations in their respective *visit()* methods.

One of these *visit()* methods of a concrete visitor can be thought of as a method not of a single class, but rather a method of a pair of classes: the concrete visitor and the particular element class. Thus the visitor pattern simulates double dispatch in a conventional single-dispatch object-oriented language such as Java.

The visitor pattern also specifies how iteration occurs over the object structure. In the simplest version, where each algorithm needs to iterate in the same way, the *accept()* method of a container element, in addition to calling back the *visit()* method of the visitor, also passes the visitor object to the *accept()* method of all its constituent child elements.

Because the visitor object has one principal function (manifested in a plurality of specialized methods) and that function is called *visit()*, the visitor can be readily identified as a potential function object. Likewise, the *accept()* function can be identified as a function applicator, a mapper, which knows how to traverse a particular type of object and apply a function to its elements.

4.4.2.2.2 Usage of JSQL parser in the system

To obtain required information from the SQL query parsed by JSQL parser, it is needed to implement appropriate visitors (according to visitor pattern's organization) that visit classes in the hierarchy which contain the information we need. Moreover, since we have specified the format of the expected input file, we do not need to implement all methods of the visitor classes but just the one's that obtain the subset of information defined by input file format. Basically, our main goal here is to transform the information from JSQL format of storing data (according to its Java class hierarchy) to our format according to our system design.

First of all, we need to create an instance of **CCJSqlParserManager** class and then to invoke its method *parse()*. If *parse()* method returns instance of **PlainSelect** class or **Union** class then we go further on extracting the information from this class. **PlainSelect** class is a class that keeps information about one SQL query without union, while **Union** class stores information about several **PlainSelect** classes that are related by union. Further on we extract information (that is needed according to Section 4.4.3) from these classes by visitors that we explain in next passages.

From the SELECT part of an SQL query we extract two lists, a list of all table attributes and a list of aggregation functions that appear. For this task we create **SelectPartVisitor** class that implements two Visitor interfaces, **SelectItemVisitor** and **ExpressionVisitor**. From the **SelectIemVisitor** we implement the method *visit(SelectExpressionItem selectExpressionItem)* for visiting one **SelectExtpressionItem** because the instances of table attributes and aggregation function that we need are contained in this class. This method reaches for the **Expression** class instance that is class attribute of the **SelectExtpressionItem** and forwards it reference to the same **SelectPartVisitor** instance (reference "this" in java) by calling its *accept()* method. Then finally methods *visit(Function aggregationFunction)* or *visit(Column column)* that are part of the **ExpressionVisior** interface are called to obtain needed information about aggregation function or table attribute respectively and information. Our implementation adds this information to the corresponding list.

From the FROM part of an SQL query we extract the list of tables that appear. For this task we create **FromPartVisitor** class that implements **FromItemVisitor** interface. We

implement two methods of this interface, *visit(Table table)* and *visit(SubJoin sj)*. First method extracts name of the table and puts it into a list. Second method, *visit(SubJoin sj)*, is for the case when we have more than one table. The JSQL parser the case when we have "table1, table2" stores as a Join class marked as simple. Then we extract the left and right part of that class and forward them again the same **FromPartVisitor** reference until we reach the tables that we then visit with *visit(Table table)* method that we have previously explained.

From the WHERE part of an SQL query we extract two list, list of joins between the tables and a list of predicates (**Clause** class instances). For this task we create **WherePartVisitor** that implements **ExpressionVisitor** interface because everything that is found in WHERE part of the SQL query is considered as one large complex expression. Then by implementing appropriate methods we divide this complex expression bit by bit until we reach the expressions that can be either joins between tables or clauses over some table attribute. Methods *visit(AndExpression expression)* is recursively called and in each call it divides complex expression to the left and right part of a and-expression until we reach the expressions that are either joins between the tables, either predicates. When it reaches join or predicate, then it invokes appropriate visit that can be: *visit(EqualsTo expression), visit(GreaterThan expression), visit(GreaterThanEquals expression), visit(MinorThan expression), visit(MinorThanEquals expression), visit(NotEqualsTo expression)*. Method *visit(EqualsTo expression)* extracts information either about joins if both arguments of equal expression are table attributes, either predicate equal to. All other methods visit corresponding predicates. Additionally, we implement two important methods used in previous ones, one for checking if a expression is a join and other that checks if a method is a simple predicate (clause).

From the GROUP BY part of an SQL query we extract a list of table attributes. For this task we create GroupByVisitor class which does not implement any visitor but since it does similar task as previous ones we named it that way. This class has a method *visitGroupBy(List list)* which goes through the list of table attributes obtained from JSQL parser and stores it in structure needed by the system.

Last class that is mentioned in this section is **InformationManager**. It represents a centralized point which uses all previous classes for extracting information from a SQL query (by using JSQ parser) and stores their output. In its constructor **InformationManager** gets the **PlainSelect** class instance that it processes by calling class's methods for extracting information about each part of the SQL query that we are interested in. These methods using above presented visitor classes for achieving their task. Than afterward we get all the information for further processing and analysis from **InformationManager**. Note that **InformationManager** is used

only for **PlainSelect** and if we have a **Union** than we need to have one **InformationManager** for each **PlainSelect** of the **Union**.

4.4.2.3 Visual Paradigm for UML

Since the UML was used for the modeling of this system some UML tool was obviously needed. Visual Paradigm for UML (VP-UML) is a UML CASE Tool supporting UML 2, SysML and Business Process Modeling Notation (BPMN) from the Object Management Group (OMG). In addition to modeling support, it provides report generation and code engineering capabilities including code generation. It can reverse engineer diagrams from code, and provide round-trip engineering for various programming languages. Its Community Edition is free made for non-commercial use and as so it was used for most of the UML diagrams in the document.

4.4.2.4 JGraph

JGraph is a Java graph visualization library. It is an open source Swing component that we used for visualizing data structures in our project. Nice layouts for automatically positioning of the diagrams are provided. One of them is a layout for tree structure which is considered as a special kind of graph (graph that has certain restrictions). Because of this, JGraph very much corresponded to the needs of MAC tree visualization in our system.

4.4.3 Coding

Design of the system we have already seen in the Section 4.3.2 and here we focus on some important implementation parts specific for the framework. The goal of this section is to give an idea of how system works in a little more detailed way. We outline the methods that contain main system's logic.

Before going to the explanations of the methods we need to note one structural detail. In this implementation the tree representing MAC is implemented as n-ary tree. This solution is more flexible and better fits the way how JSQL parser is working. It should be easy in the future to restrict the number of nodes to only two if needed.

After we have earlier seen design of the system with special emphasis on the data structures for storing information about MAC and also how we use JSQL parser for extracting information from an SQL query now we show how to connect these two points. We explain how to exploit the information about parsed queries to identify multidimensional operators plus raw

data and then build MAC. Identifying of the multidimensional operators is done in several methods of **MAC** class:

- *rawDataIdentificaton()* – this method identifies **RawDataElement** by a list of tables from the FROM part of a SQL query obtained from **InformationManager** instance.
- *rollUpIdentification()* – this method takes next input parameters: current Node of the MAC, list of Group By table attributes, list of functions from the SELECT part of a SQL query and a list of table names of FROM part of a SQL query. With these input parameters it identifies **RollUpElement**s as shown in Figure 17. If the GROUP BY part of a SQL query exists it means that all dimension level identifiers will be included there and that there are certainly related aggregation functions in the SELECT part. Otherwise, without GROUP BY part of a SQL query, we can still have Roll up operator identified if there aggregation functions exist in the SELECT part.

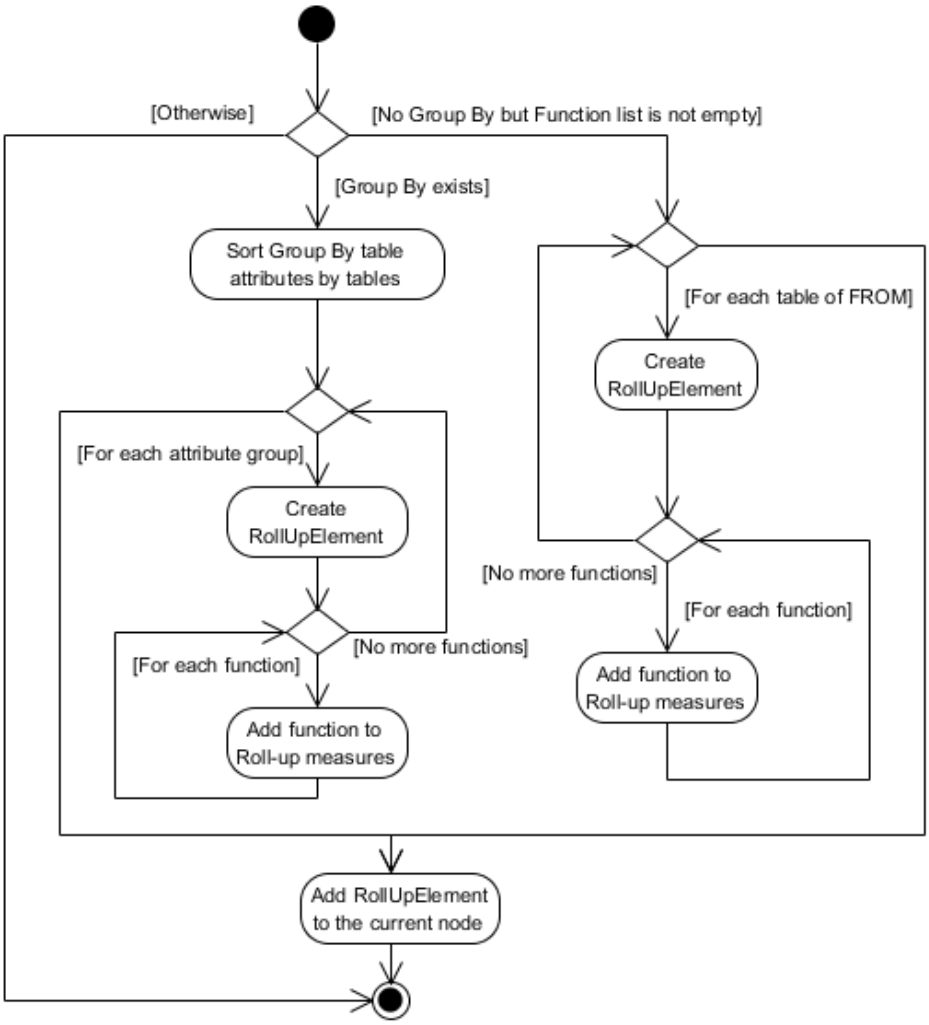

Figure 17. Roll-up identification

- *changeBaseIdentification()* – this method takes next input parameters: current Node of the MAC, list of Group By table attributes and a list of table attributes from the SELECT part of a SQL query. With these input parameters it identifies **ChangeBaseElement**s as shown in Figure 18. Important characteristic to notice is that if we have **RollUpElements** identified before **ChangeBaseElement** we know that table attributes to which we are rolling up (of the **RollUpElements)** need to be present in the table attributes of the from base of **ChangeBaseElement**.

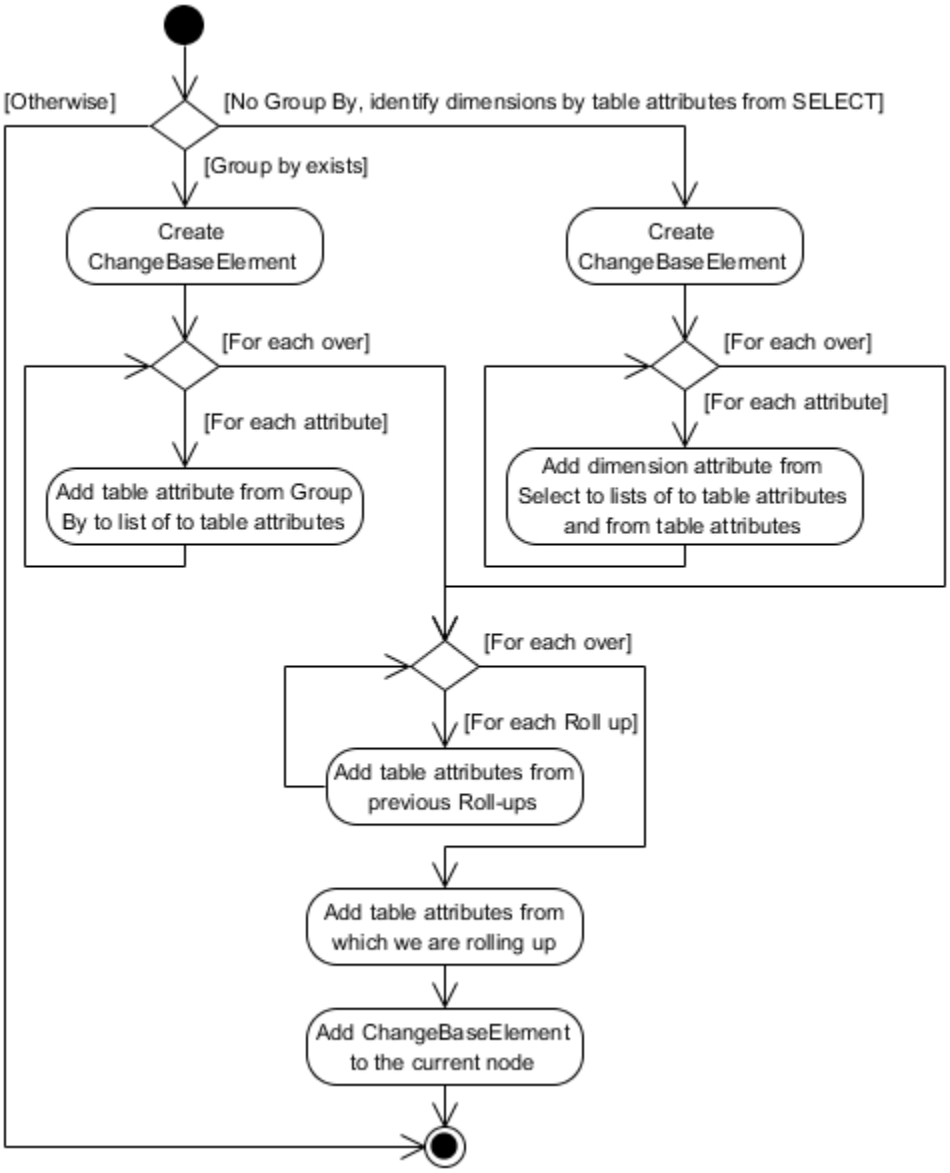

Figure 18. Change Base identification

- *selectionIdentification()* – this method takes next input parameters: current Node of the MAC, list of clauses from WHERE part of a SQL query and a list of table

attributes from the SELECT part of a SQL query. With these input parameters it identifies **SelectionElement**s as shown in Figure 19.

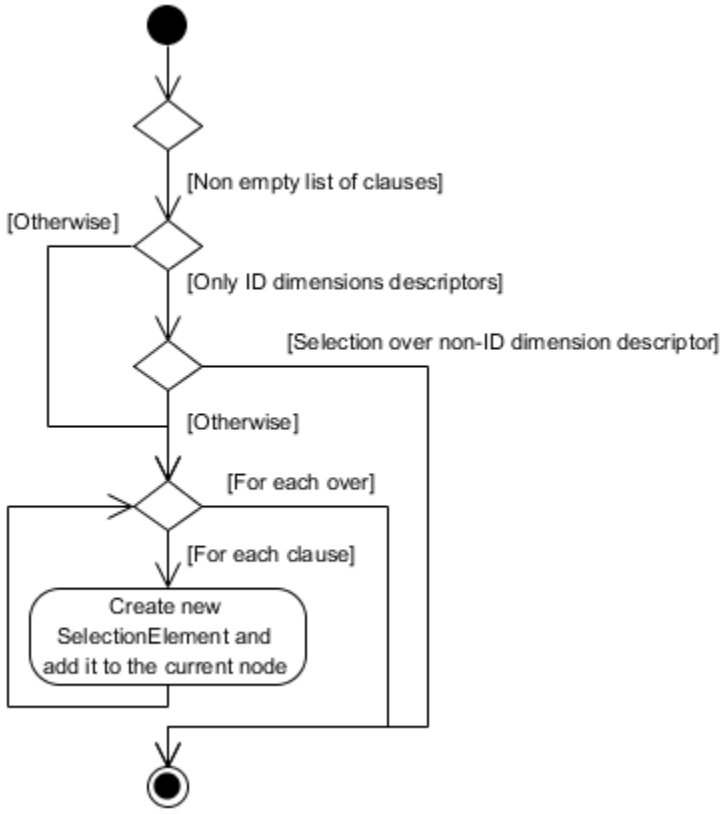

Figure 19. Selection identification

- *projectionAndDrillAcrossIdentification()* – this method takes next input parameters: current Node of the MAC, list of functions from the SELECT part of a SQL query and a list of table attributes from the SELECT part of a SQL query. With these input parameters it identifies **ProjectionElement**s and **DrillAcrossElement** as shown in Figure 20. These to types of elements are identified together because **DrillAcrossElement** is identified if we have more than one **ProjectionElement**s, while one **ProjectionElement** is contains measures belonging to the same table.

Figure 20. Projection and Drill Across identification

By using all these method we extract MAC in **MAC** class's method *extractQueryToMAC()*. This method first creates **InformationManager** (for details turn to Section 4.4.2.2.2) and then uses information that it provides to parameterize and invoke above mentioned methods. As we saw all methods except the last one add **UnaryElements** to the current node. The method *projectionAndDrillAcrossIdentification()* creates new node because of n-ary DrillAcross node. Further on, in method *extractQueryToMAC()* in case of a union it creates new n-ary Union node with corresponding children nodes that are actually queries parsed by the same method in a recursive call. This way, the tree MAC structure is built.

When MAC of a SQL query is extracted the next step is to normalize it. Since it is an important process in this system we make a general overview of it with more detail illustrations in the form of action diagrams or pseudo code when needed. The steps of the normalization process are:

- First, *normalizeMAC()* method of **MAC** class is invoked. This is a simple method that invokes the root node's method *normalizeAllNodes()*.
- Method *normalizeAllNodes()* make a postorder traversal of all its child nodes and invokes their *normalizeNode()* method:

  normalizeAllNodes()

  Begin
  if node has children nodes
  for each child node
  normalizeAllNodes()
  end_for
  end_if
  normalizeNode()
  End
- Method *normalizeNode()* invoked from *normalizeAllNodes()* is shown in Figure 21.
- Method *normalizeNode()* invokes several methods to do the need checkups before moving elements. Methods that need to be pointed out are *normalizeNP()* (we have the invoke of this method in Figure 21) and method *switchElements()* that is invoked from the method *normalizeNP()*.
- While the logic of normalizing n-ary nodes is mainly concentrated in the method *normalizeNode()* itself, normalizing of the unary nodes is placed in the method *normalizeNP()* that is invoked from *normalizeNode()* method. Method *normalizeNP()* goes through the list of node's elements from the root-side to leaf-side and switch elements places if possible according to the Table 5. This switching of elements places is done by invoke of *switchElements()* method. However one specific case important to mention is when this method tries to switch places of two **ProjectionElements**. In addition to not exchanging their places (check Table 5) it will remove leaf-side **ProjectionElement** since it does not make sense to have two **ProjectionElements** in one NP of a NMAC.
- Beside just the routine switching of elements places, method *switchElements()* does one more important action. When it switch place of a **RollUpElement** and **ChangeBaseElement** it changes the schema of the **ChangeBaseElement** to make it possible for these two elements to switch their places. For this change both elements

need to deal with same dimensions. In case of moving **RollUpElement** over **ChangeBaseElement** it changes all the attributes of same dimension in the **ChangeBaseElement** with the input attribute(s) of same dimension belonging to the **RollUpElmenet**. In case of moving **ChangeBaseElement** over **RollUpElement** it changes all the attributes of same dimension in the **ChangeBaseElement** with the output attribute(s) of same dimension belonging to the **RollUpElmenet**.

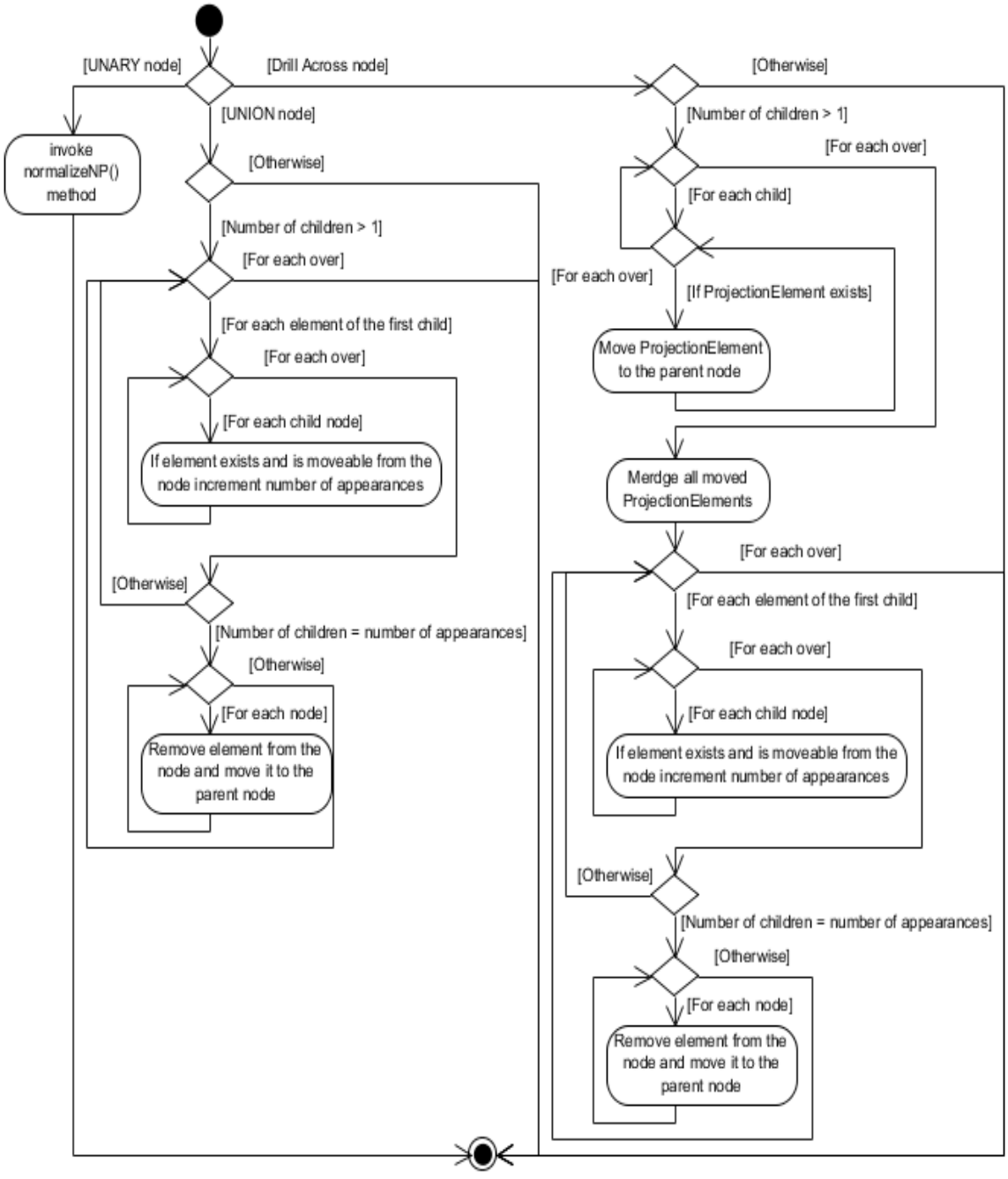

Figure 21. Method *normalizeNode()*

One more feature of this system that is important for future upgrades is possibility to get data cube after each multidimensional operator effect. All **UnaryElement**s that represent multidimensional unary operators together with raw data element implement method *getCube()* that returns the current data cube after the effect of the multidimensional operator or output of the raw data element. Multidimensional unary operators as an input take previous data cube, while raw data element does not need any input. The effect of the method according to the element it belongs to is:

- RawDataElement – it return the list of primary keys of all tables contained.
- ProjectionElement – it removes all the projected measures from the input data cube and adds its measures.
- RollUpElement – it removes all table attributes that represent identifier of the level from which it rolls up and adds all table attributes that represent identifier of the level to which it rolls up.
- SelectionElement – it does not changes data cube
- ChangeBaseElement – it removes identifiers of previous dimensions and adds new dimension identifiers.

To obtain previous data cube each element, except RawDataElement, invokes its parent's node *getPreviousCubeOfNode()* method. This method depending from the type of the node and position of the element returns data cube of previous element. Obviously this method always recursively goes to the raw data elements and then step by step applies the effects of the elements in between. However there are specificities that need to be pointed out:

- If there is an element in the node previous to the current element which invokes *getPreviousCubeOfNode()* node returns the data cube returned by the previous element.
- If there is not a previous element in the node then obtaining of previous data cube depends from the node type. If the current node type is UNARY then from the first non empty offspring node it returns the root-most element's data cube. If the current node type is UNION then from the first non empty offspring node it returns the root-most element's data cube but with the check if all the children nodes return the same data cube. If the current node type is DRILL_ACROSS then it returns the data cube made by merging data cubes of all the children in data cube which has the dimensions of one data cube (note that these dimensions have to be the same for all the children's data cubes) and the measures coming from all children's data cubes. Before doing this, check if all the children's data cubes are compatible is done.

4.4.4 Obstacles and solutions

While working on a research project we usually encounter some problems and obstacles. These issues that are technical or topic related affect our research process in timing, feasibility, quality, cost and many other aspects. As such, they are integral part of a project and should be mentioned and discussed. While cost aspect we analyze in a chapter dedicated especially for it, here we present the rest of the issues together with the corresponding solutions.

First obstacle encountered was finding a good SQL parser. There were three SQL parsers that we considered: DTP parser, General SQL parser and JSQL parser. Out of all these options DTP parser seemed as a most serious and powerful solution. Project leader for the platform to whom DTP parser belongs to is a person working for IBM so that fact seemed really promising. The documentation was really poor but still we decided to try to use it. This parser is a plug-in for Eclipse platform and this required additional time needed for first getting acquainted with this platform. After becoming familiar with the platform the setting of the plug-in with this parser was easy. However, including of the parser into a project happened to be very much undefined procedure. There was no option of including plug-in into project and no help or some other documentation or example of how to use it. This was solved by including all the files from the plug-in folder but then new problem arise. Still some .jar file was reported missing and after many hours of web search instructions how to solve this were found on some forum. Finally everything seemed as ready for use, even some UML documentation of the structures used by parser was found, but it turned out that a parser cannot be used for our project. It did not enable access to the data structures where it keeps information about the parsed queries. This switched our attention to the other two remaining parsers. The General SQL parser worked really nice but since it is a commercial product it had some limitations for non-commercial use so we could not use it either. The last option was JSQL parser which had some note that it is slow at first parsing and we needed I fast parser so this was not promising solution. However, it turned out to be fine and efficient parser and it is in detail presented in Section 4.4.2.2. It is possible to use it with NetBeans IDE with which was very convenient for us so the problem was finally solved although a time overhead was again needed for acquitting with this last chosen parser. That is the main reason for an exhaustive explanation of the parser and its usage in Section 4.4.2.2.

After this technical obstacle that is not directly related to the thesis but more to the prerequisites for working on the thesis, now we present important obstacles emerged while working on the system.

Main obstacle while developing this system was absence of the information about the database schema. We only had the SQL query text and information it contains. This brought out

the problem of distinguishing dimensions from measures, and also of distinguishing id descriptors from non-id descriptors. However, there are certain rules for a SQL query that helped us deal with this issue. First, if the SQL query has a GROUP BY part it enables us to identify all the id descriptors of the dimensions because they have to be included in it. Also, in that case measures will be aggregated in the SELECT part of the SQL query. When the query does not contain the GROUP BY part then we introduced the assumption that all the measures will have the "M_" prefix before the name of the attribute. These rules enabled us to overcome these problems. Nevertheless there were still some open issues coming from this absence of database schema.

Next issue was the lack of information about the universal relation which consists of all the tables of the database schema and also the lack of information about the primary keys of the tables. This situation made problems for identifying the multidimensional operators. The solution for this issue was introduction of so-called "phantom" values. This "phantom" value was used to represent primary key of the table like for example "table_name.phantom" where it marks the primary key of the table with "table_name". Also, when we needed to use a universal relation of all tables, since we did not know all included tables due to a lack of database schema, we introduced "phantom.phantom" value for simulating it. These solutions enabled further work on the system. In future, with the introduction of the database schema, these phantom values will be changed with appropriate values.

The last solution found for solving the problem of the absence of the database schema information was that it is required to have table name alias in front of the table attribute name (which is sometimes the case even when the schema information is considered). Again, this requirement comes from the process of identifying the multidimensional operators in SQL queries. This enables us to detect the table of the table attribute even without the information about the database schema.

New obstacle that arose while working on the normalization of the MAC was a situation when Change Base operator and Roll up operator could not exchange places because of structural conflict even though they should be able according to their meanings. Solution for this situation was change of the operator schema of the Change Base operator so it would not get in conflict with Roll up operator and at the same time would not change the final outcome of the application of both operators. Details about this procedure can be found in Section 4.4.3.

Many more difficulties arose when dealing with identification of multidimensional operators but it would not make sense to list them all here. The solutions to these problems we have had the chance to see in previous sections as the final system realization. Normalization also

brought equally numerous issues and the solutions are also already provided as the final system realization.

All these obstacles illustrate the overall complexity of the process of MAC identification and normalization. They helped in even better understanding and improvement of the framework. It seems that from this point on process of query recommendations would be something clear and defined. By obtaining NMAC from the queries we managed to extract essential information about multidimensional operators in queries and query sessions and store that information in a structured and organized way so it may be further exploited for various use.

4.5 Testing

Important phase in every software project certainly is testing. For the testing of the system various testing techniques can be used. According to [23], these test techniques include, but are not limited to, the process of executing a program application with the intent of finding errors or other defects.

Software testing can also be stated as the process of validating and verifying that a software program:

- meets the business and technical requirements that guided its design and development,
- works as expected and
- can be implemented with the same characteristics.

Software testing can be implemented at any time in the development process. However, most of the test effort occurs after the requirements have been defined and coding process has been completed. As such, the methodology of the test is governed by the software development methodology adopted.

A primary purpose of testing is to detect software failures so that defects may be discovered and corrected. This is a non-trivial pursuit. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions. The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. In current culture of software development, a testing organization may be separate from the development team.

There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed.

Considering that for the testing of this system we use both white box techniques since the testing is done by the developer who already knows the systems algorithms and data structures, but the test are conducted by the principles of black box techniques – for a given input check the output, we can say that we used grey box testing. Grey box testing involves having knowledge of internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level.

Regarding to the testing levels present levels are unit testing, integration testing and final system testing depending from the stage of development.

Unit testing, also called component testing refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors. In the case of this system unit testing is conducted to check the classes (visitor classes) needed for the work with JSQL parser. For the input .sql files, it is checked whether those classes extract needed information. These classes are basically independent of each other so this is appropriate testing.

As further developing the system, besides unit testing (if possible) of new classes integration testing is necessary since the classes now depend one from another. Integration testing works to expose defects in the interfaces and interaction between integrated components. Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

Finally, system testing is conducted as testing of completely integrated system to verify that it meets its requirements. For this system requirements can be found in Section 4.2.

Test cases used for the testing of the systems are designed in gradual order. Input .sql files must comply with expected input format presented in Section 4.4.3. and respect the syntax rules of SQL language (for example, if we have a GROUP BY clause, we must have at least on aggregation function in SELECT part of a SQL query and all other arguments in that SELECT PART except aggregation function(s) must be included in GROUP BY clause and similar rules). By gradual order of the test cases we consider gradual including of multidimensional operators in the tests, first unary and then binary multidimensional operators (for example, Union multidimensional operator we include the last in the test cases). After gradual inclusion of all multidimensional operators, next important step in creating is mixture of different multidimensional operators and their order. This is especially important for the testing of the

normalization algorithm. This arranged set of test cases used in previously explained way represents chosen technique for testing of the system. General organization of testing process is illustrated in Figure 22.

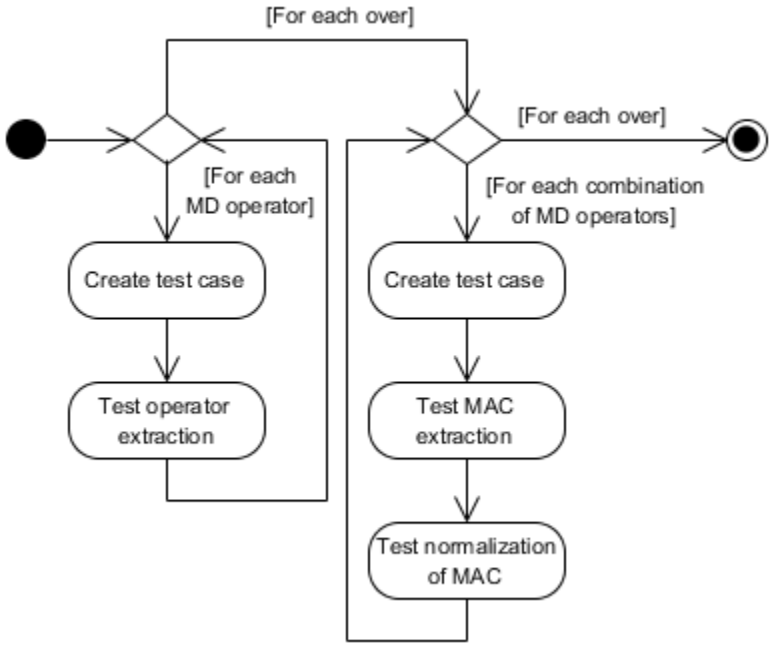

Figure 22. Testing process illustration

5 User manual and Demo

This chapter presents the guide for usage of the application's GUI. At the same time, this chapter represents the system's demo since it is based on a typical SQL query and all the functionalities of the system are presented.

For the running example in this chapter, the input file "Union.sql" contains next sql query:

```
SELECT date_dim.d_month_seq, customer_dim.state, SUM(catalog_sales.cs_quantity)
FROM catalog_sales, date_dim, customer_dim
WHERE catalog_sales.cs_sold_date_sk = date_dim.d_date_ski AND
        catalog_sales.cs_customer_id = customer_dim.c_customer_id AND
        date_dim.d_month_seq = 'january' AND
        customer_dim.state = 'Spain'
GROUP BY date_dim.d_month_seq, customer_dim.state
UNION
SELECT date_dim.d_month_seq, customer_dim.state, SUM(catalog_sales.cs_quantity)
FROM catalog_sales, date_dim, customer_dim
WHERE catalog_sales.cs_sold_date_sk = date_dim.d_date_ski AND
        catalog_sales.cs_customer_id = customer_dim.c_customer_id AND
        date_dim.d_month_seq = 'january' AND
        customer_dim.state = 'Sweden'
GROUP BY date_dim.d_month_seq, customer_dim.state;
```

This query retrieves total quantity of items sold by catalog sale, for month January and countries Spain and Sweden.

In Figure 22, we see a start up screen that appears when starting the application.

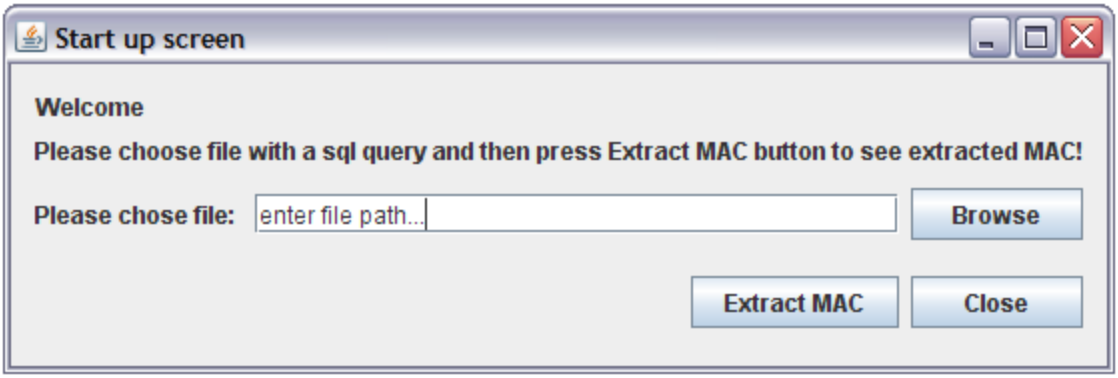

Figure 23. Start up screen

In start up screen user should choose the .sql file that contains SQL query for which she wants to extract MAC. This can be done either by textual entering the file's absolute path, either by clicking on a "Browse" button. When clicking on a "Browse" button screen in Figure 23 opens.

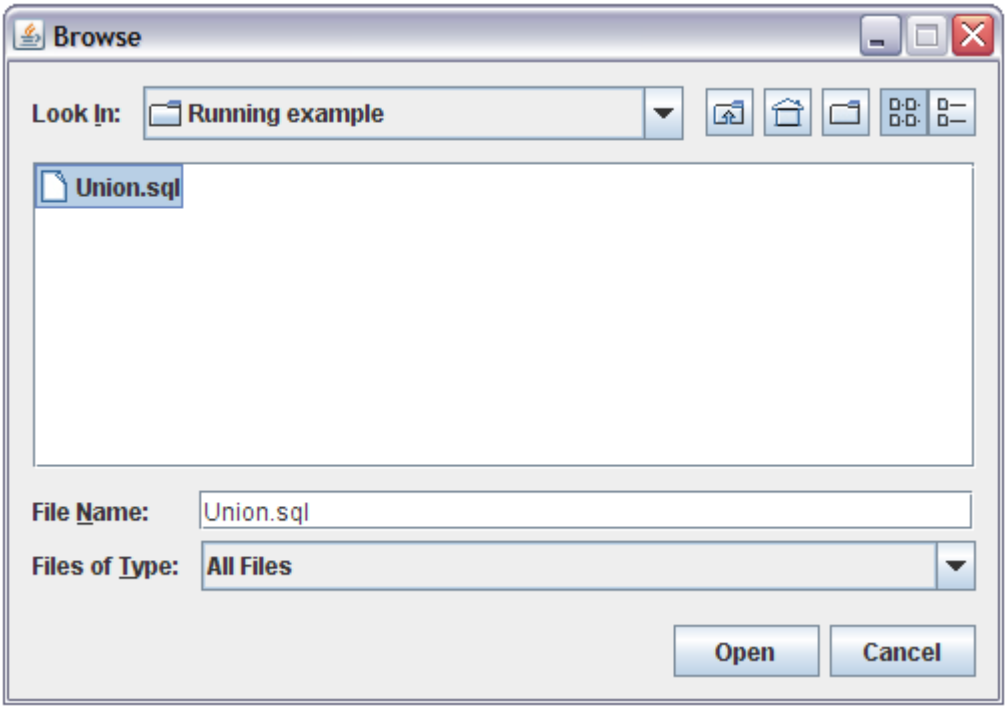

Figure 24. Browse screen

Now, user can chose the file and click on button "Open". Next, after choosing file in one of the two possible ways user should press button "Extract MAC" shown in Figure 22. This press of a button opens a new screen shown in the Figure 24. User can resize the screen to fits the size of the tree, or use the scroll bars for seeing different parts of it. Regarding to the demonstration part, here we can see the extracted MAC for a running example query.

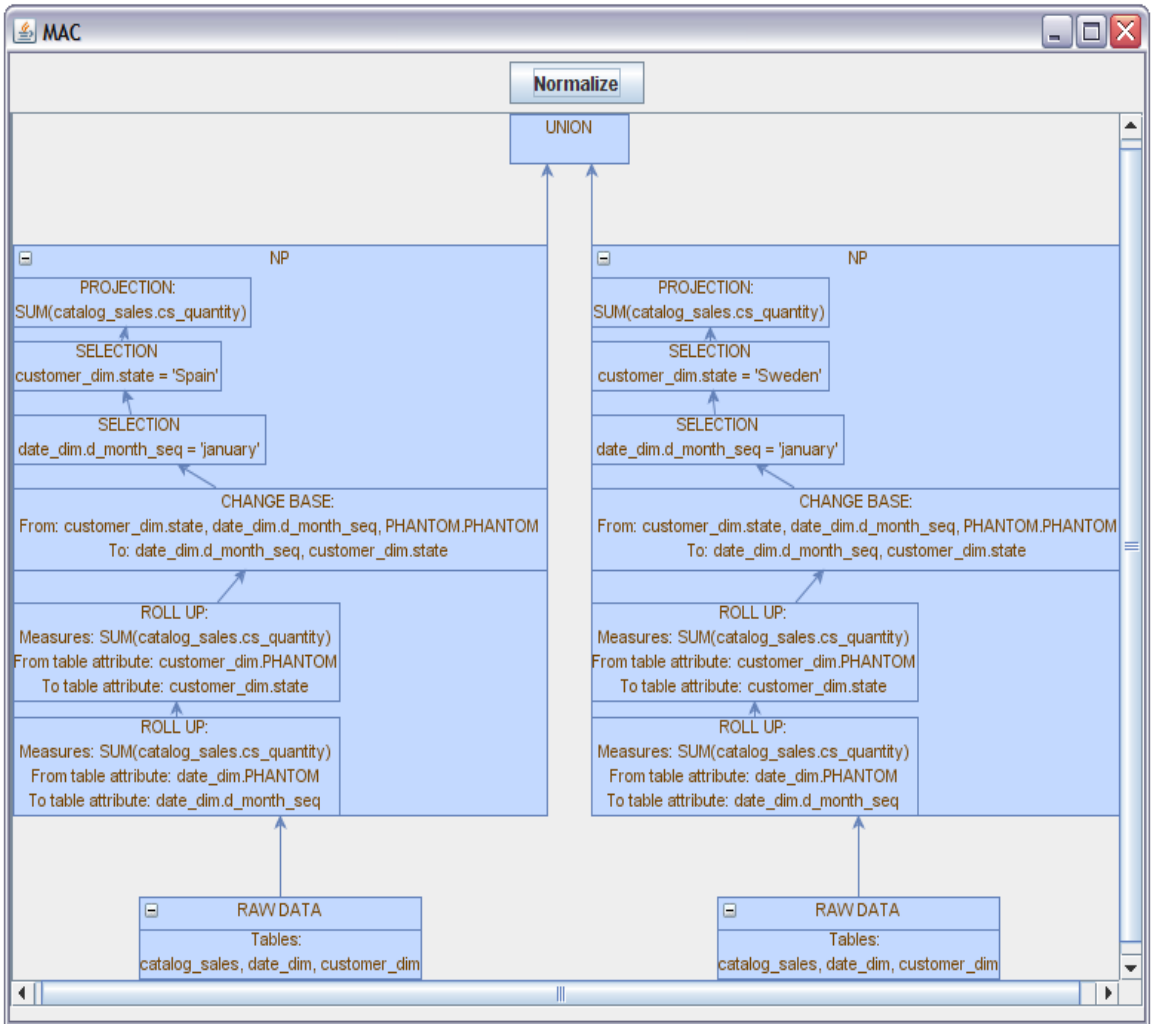

Figure 25. MAC screen

After seeing the visualization of the extracted MAC, user is able to also see visualization of the same MAC after normalization. For this, user needs to click on the button “Normalize” shown in Figure 24. Visualization of normalized MAC or NMAC is shown in Figure 25. Regarding to the demonstration part, here we can see MAC after normalization for a running example query.

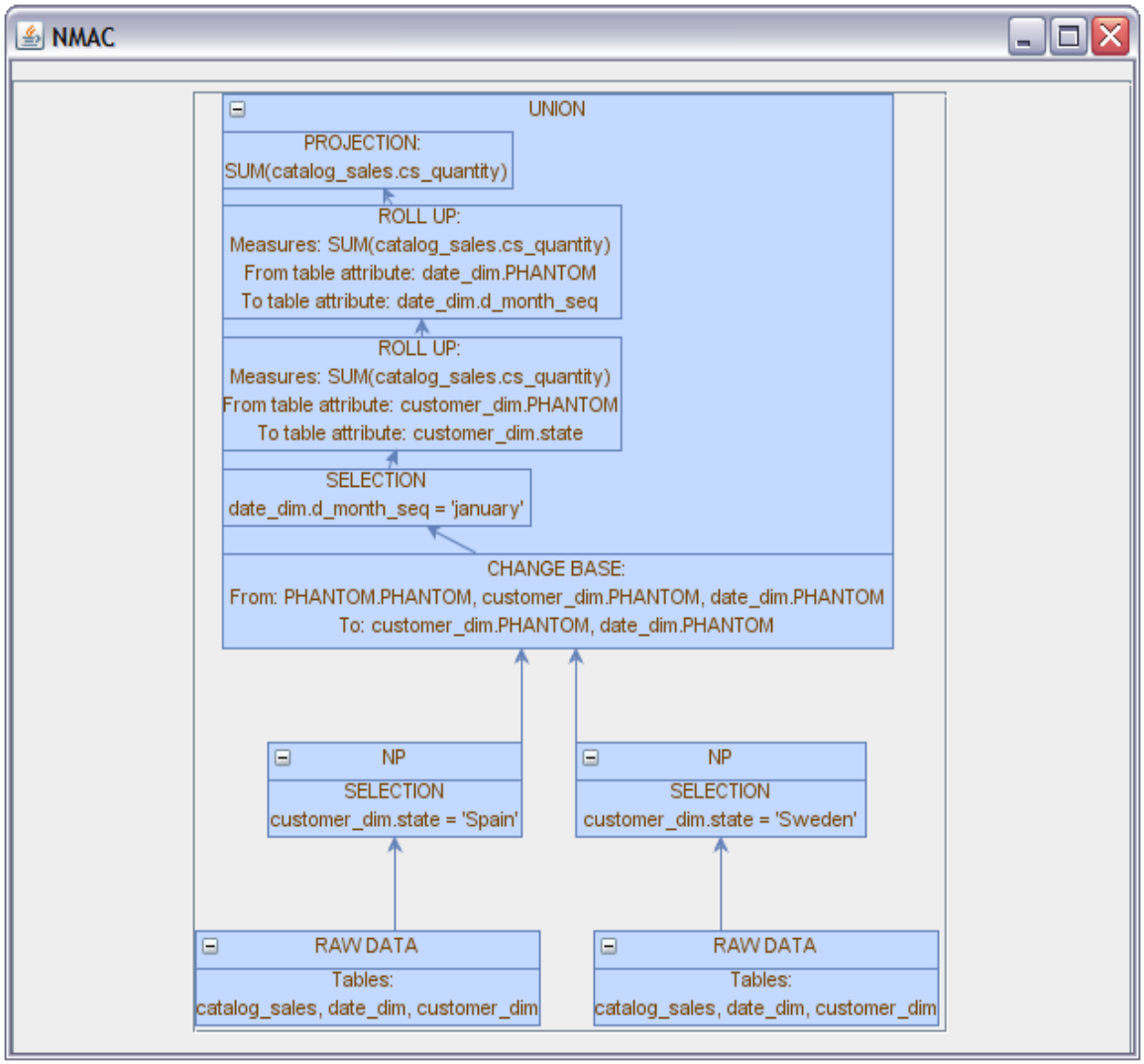

Figure 26. NMAC screen

As we earlier sad, this system is designed as a batch process so this GUI is just for the presentation purposes to illustrate the results.

6 Project costs

This chapter presents the project cost. To provide through discussion about the cost of the project we shall first analyze the initial plan of the project. Then right after we present final project organization. From these two perspectives we assess the outcome of the project through the prism of project costs. Since this is a research project, phases are specific and divided between research (mostly theoretical) part and implementation (mostly practical) part. One more particularity of research process is that some phases are more complex than expected and due to that fax initial and final plan of the project differ a lot as it was the case in this project.

6.1 Initial project plan with estimated costs

As already said, first we present the initial project plan. This plan consists of next several phases that were planed according to the theoretical basis of this project:

- State of the Art – this phase consists of research process focused on the most important approaches currently existing related to the domain of our work.
- New approach – this phase consists of research process focused on the new approach presented in this thesis.
- Technical prerequisites – this phase consists of search for and familiarization with the new tools required for the development.
- Prototyping – this phase considered the implementation of a system that had three functionalities: identifying of MAC, normalizing of MAC, bridging of MACs. As usually this phase consisted of sub phases:
    - Requirements analysis
    - Design
    - Implementation
    - Testing
- Documentation – this phase consists of documenting of whole process and all the results and phases.

Corresponding to this organization of project phases expected time scheduling is presented in Table 8. Project duration was five months.

Table 8. Initial time plan

Phase type	Phase	Time distribution
Theoretical	State of the Art	160 hours
	New approach	160 hours
Practical	Technical prerequisites	70 hours
	Requirements analysis	35 hours
	Design	35 hours
	Implementation	175 hours
	Testing	35 hours
	Documentation	70 hours
Total		740 hours

When assessing the costs of the project according to this plan we consider costs of human resources i.e. job roles in this process and non-human resources. Estimated costs are presented in Table 9 and Table 10 respectively.

Table 9. Planed human resources costs

Phase	Job position	Working hours	Cost
State of the Art	Researcher	160	160 * 30 = 4800
New approach	Researcher	160	160 * 30 = 4800
Technical prerequisites	Developer	70	70 * 20 = 1400
Requirements analysis	Project manager	35	35 * 40 = 1400
Design	System analyst	30	35 * 30 = 1050
Implementation	Developer	175	175 * 20 = 3500
Testing	Tester	35	35 * 15 = 525
Documentation	Researcher	70	70 * 30 = 2100
Total			19575 €

Table 10. Non-human resources costs

Resource	Cost
Notebook: Toshiba Satellite L655-1JK Intel-Core i5-480M 2.66GHz 4096MB 640GB Blu-Ray 15.6" WXGA Win7P	850
Microsoft Office 2010	150
NetBeans IDE 6.9.1	0
JSQL parser	0
Visual Paradigm for UML 8.2 Modeler Edition	100
Total	1100 €

6.2 Final project plan with estimated costs

Final project plan and costs are presented in this section. Regarding to the phases of the project they remained the same except the prototyping phase which does not include bridging sub phase. This change is caused by changes in the time planning as shown in Table 11.

Table 11. Final time plan

Phase type	Phase	Time distribution
Theoretical	State of the Art	160 hours
	New approach	**180** hours
Practical	Technical prerequisites	**140** hours
	Requirements analysis	**25** hours
	Design	**25** hours
	Implementation	**200** hours
	Testing	35 hours
	Documentation	70 hours
Total		835 hours

While the cost of non-human resources have not changed and have been as planed, the human resources costs changed according to the changes in time planning (changed values are bolded in Table 11). This changed final human resources costs are shown in Table 12.

Table 12. Final human resources costs

Phase	Job position	Working hours	Cost
State of the Art	Researcher	160	160 * 30 = 4800
New approach	Researcher	160	**180** * 30 = **5400**
Technical prerequisites	Developer	70	**140** * 20 = **2800**
Requirements analysis	Project manager	35	**25** * 40 = **1000**
Design	System analyst	30	**25** * 30 = **750**
Implementation	Developer	175	**200** * 20 = **4000**
Testing	Tester	35	35 * 15 = 525
Documentation	Researcher	70	70 * 30 = 2100
Total			21375 €

6.3 Discussion

As it is obvious, the initial plans and final plans differ. Main reasons for this are difficulties with finding suitable technical prerequisites (concretely SQL parser) and higher complexity of the planed system functionalities. Realized functionalities required more time as it can be noticed from the tables. Main obstacles that caused these deviations were already mentioned in Section 4.4.4. Less time was needed for the requirements and design but more for the implementation. Summarized cost comparison we can find in Table 13.

Table 13. Comparison of initial and final costs

	Human resources	Non-human resources	Total
Initial costs	19575 €	1100 €	20675 €
Final costs	21375 €	1100 €	22475 €
Difference	1800 €	0	1800 €

As obvious, final plan required more time (and costs) then it was planed. This can be justified by unpredictability of research process in general. Final project cost more for 8,7% from the initial planed price.

7 Conclusions

In this master thesis we have presented new approach for representing SQL queries and query sessions in systematic, structured and normalized way. The starting aim of this thesis was to reach the point of using this approach for generating query recommendations, however it turned out that the task of representing query session itself is complex enough. Theoretical framework standing as a basis of this approach is a sound foundation for practical work. Nevertheless dealing with the analysis of the SQL queries for extracting the multidimensional operators happened to be tedious task that required high efforts. Therefore realization of this work is a great achievement since it opens many new possibilities for further development and exploitation including query recommendation.

In Chapter 2, we have shown that current approaches in this field do not consider the semantics of interconnection between the queries of the same session and that is so if they consider interconnection at all. In those, rare cases this analysis is based on consideration of database tuples retrieved by the query. This analysis lacks first on performance feasibility since it deals with enormous data sets and more important, it lack in semantics considerations since two very similar queries with different filtering condition may retrieve two disjoint sets of data. This deficiency was tried to be compensated by analyzing query fragments and in such a way detect similarity between queries even if they retrieve different data but again it lacks of semantics because it is based on mainly syntactical analysis. It is well known that by using SQL two same queries can be written in two different ways. Due to this current situation in this field, new approach that is presented brings innovations for tackling this issue.

Chapter 3 presents this new framework. In today's world that is getting more and more digitalized large decisional databases that capture large part of that digitalization process represent new resources of modern society. Exploration of this wealth of data can bring great benefits and useful information. However, navigating and exploring these huge decisional data repositories is not routine neither easy task. Obliviously there is a need for capturing this knowledge of significant and interesting spots in this "colorful picture" of data. The most intuitive way for achieving this is capturing the information about the previous searches of users. This brings us to the importance of capturing the user sessions that is analyzed in thesis. Its theoretical part as a novel framework represents backbone of later developed system. Possibility of extracting multidimensional operators out of analytical queries represents a powerful way for

gaining precious information about user behavior and also interesting parts of the decisional database. Furthermore when these multidimensional operators are organized in a structured way such as MAC and even more NMAC possibilities for exploitation widely extend. This is something really worth of working on without emphasizing.

Chapter 4 gives one possible implementation of framework presented and by itself it supports the theory with practical evidences. However, often it is not easy and straight forward to translate theory into practice, neither was it this time. This was a challenging task but worth of every effort. There were many obstacles due to the complexity of all included settings. Solving these issues gives even more value to the developed system and testifies of framework feasibility. Developing of such system is usually entrusted to a big development teams in large companies so being able to contribute in these small but valuable way represents a great honor.

Chapter 5 represents a demonstration of developed system and illustration of the process "under the hub". At the same time it gives sufficient instructions on how to use system interface and become familiar with it in just a short notice.

Chapter 6 presents us a potentially costs of this project and illustrates difficulties occurred in a way of differences between initial and final plan. This overview gives a bit wider perspective on this systems development and work on this master thesis in general.

7.1 General conclusions

This master thesis presents a promising novel approach for storing information about SQL queries and query session. Throughout the thesis we tended to justify this statement. Query recommendations are one possible way of exploiting information obtained by MACs and NMACs. Storing NMACs of all queries posed in the system gives us concise set of information about the usage of the system and enables us efficient handling of that information. In the context of query recommendations, when a user poses a query the system should be able to detect the similarity between current and previous queries based on the MACs and recommend similar queries. Furthermore, depending from the recommendation algorithm system can even suggest queries never posted based on the analysis of existing queries which can be important when searching the decisional database for some unexpected information. For example, this can be achieved by recommending new operators to be considered instead of queries. Recommender systems are already very well developed in other branches of computer systems (such as web applications) that use algorithms that could also be applied for decisional database system's

settings. Definitely, one thing is firm, that this approach brings freshness and novelties into the emerging area of analysis of decisional databases.

7.2 Future work

The approach presented in this master thesis brings many promising and challenging possibilities for future work. Beside already mentioned query recommendation this way of storing queries can be exploited for query personalization by enabling the identification of user's interest, query optimization, generating of user-oriented multidimensional databases and others.

Currently more important future work certainly relates to the presented system:

- Introducing the underlying system schema information is one of the first next upgrades with all improvements and changes it brings.
- More detailed elaboration of bridging phase is needed followed by the realization of the same.
- Analysis of input data over which the navigation is done in order to give different context to same navigation over different data.

8 Bibliography

[1] Khoussainova, N., Balazinska, M., Gatterbauer, W., Kwon, Y., Suciu, D.: A case for a collaborative query management system. In CIDR 2009: Proceedings of the 4th biennal Conference on Innovative Data Systems (2009)

[2] Marcel, P., Negre, E.: A survey of query recommendation techniques for data warehouse exploration. 7èmes journées francophones sur les entrepôts de données et l'analyse en ligne (EDA 2011), Clermont-Ferrand, Juin 2011

[3] Chatzopoulou, G., Eirinaki, M., Polyzotis, N.: Query recommendations for interactive database exploration. In SSDBM, pp. 3–18. (2009)

[4] Stefanidis, K., Drosou, M., Pitoura E.: "You May Also Like" Results in Relational Databases. In PersDB. (2009)

[5] Akbarnejad, J., Chatzopoulou, G., Eirinaki, M., Koshy, S., Mittal, S., On, D., Polyzotis, N., Swarubini Vindhiya Varman, J.: "SQL QueRIE Recommendations", PVLDB. (2010)

[6] Khoussainova, N., Kwon, Y., Balazinska, M., Suciu, D.: Snipsuggest: Context-aware autocompletion for sql. PVLDB 4(1), 22–33. (2010)

[7] Yang, X., Procopiuc, C. M., Srivastava, D.: "Recommending Join Queries via Query Log Analysis", ICDE (2009)

[8] Kimball, R., Reeves, L., Thornthwaite, W., Ross, M.: The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses. John Wiley & Sons, Inc. (1998)

[9] Romero, O., Abello, A.: On the Need of a Reference Algebra for OLAP. In: DaWaK. pp. 99{110 (2007)

[10] Abello, A., Romero, O.: On-Line Analytical Processing. In: Liu, L., • Ozsu, M.T. (eds.) Encyclopedia of Database Systems, pp. 1949{1954. Springer (2009)

[11] Abello, A., Samos, J., Saltor, F.: Y AM2 (Yet Another Multidimensional Model): An extension of UML. Information Systems 31(6), 541{567 (2006)

[12] Garcia-Molina, H., Ullman, J.D., Widom, J.: Database Systems. Prentice-Hall (2008)

[13] Lenz, H.J., Shoshani, A.: Summarizability in OLAP and Statistical Data Bases. In: Ninth Int. Conf. on Scienti_c and Statistical Database Management (SSDBM). pp. 132{143. IEEE Computer Society (1997)

[14] Weipeng P. Yan and Per-_Ake Larson: Performing Group-By before Join. In: ICDE. pp. 89{100. IEEE Computer Society (1994)

[15] Golfarelli, M., Rizzi, S.: Data Warehouse Design: Modern Principles and Methodologies. McGraw-Hill (2009)

[16] Kimball, R., Reeves, L., Thornthwaite, W., Ross, M.: The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses. John Wiley & Sons, Inc., (1998)

[17] Romero, O., Marcel, P., Abello, A., Peralta, V., Bellatreche, L.: Describing Analytitcal Sessions Using a Multidimensional Algebra. 13th International Conference on Data Warehousing and Knowledge Discovery. (2011) – to appear

[18] http://www.agilemodeling.com/

[19] http://agilemanifesto.org/

[20] http://en.wikipedia.org/wiki/Agile_software_development

[21] http://en.wikipedia.org/wiki/Batch_processing

[22] http://en.wikipedia.org/wiki/Java_(programming_language)

[23] http://en.wikipedia.org/wiki/Software_testing

[24] http://en.wikipedia.org/wiki/Requirements_analysis

9 Glossary

- Agile software development – A group of software development methods based on iterative and incremental development.
- Analytical query –All the queries that can be represented from the MD point of view.
- Business Intelligence (BI) – A broad category of applications and technologies for gathering, storing, analyzing, and providing access to data to help enterprise users make better business decisions.
- Data cube schema – It is a MD interpretation of the output produced by analytical query.
- Data structure – A particular way of storing and organizing data in a computer memory so that it can be used efficiently.
- Data warehouse – A data warehouse is a centralized repository that stores data from multiple information sources and transforms them into a common, multidimensional data model for efficient querying and analysis.
- DB – Database
- DBMS – (DataBase Management System) Software that controls the organization, storage, retrieval, security and integrity of data in a database. It accepts requests from the application and instructs the operating system to transfer the appropriate data.
- Descriptor – A level attribute of one Dimension.
- Dimension – A points of view for analysis in analytical query. Dimensions contain an aggregation hierarchy of levels representing different granularities (or levels of detail) to study data.
- Fact – A subject of analysis in analytical queries.
- GUI – Graphic User Interface
- MAC – Multidimensional Algebraic Structure
- MD – Multidimensional
- MDA – Multidimensional algebra
- Multidimensional data model – Data model that is composed of logical cubes, measures, dimensions, hierarchies, levels, and attributes.
- Measure – An attribute of one Fact.
- NMAC – Normalized Multidimensional Algebraic Structure
- NP – Navigation path
- OLAP – On-Line Analytical Processing (OLAP) is a category of software technology that enables analysts, managers and executives to gain insight into data through fast, consistent, interactive access to a wide variety of possible views of information that has been transformed from raw data to reflect the real dimensionality of the enterprise as understood by the user.
- Operator schema – Information that defines the effect of the MD operator over data cube schema(s).
- Query session – Several queries posed by the same users, expected to be mutually correlated.
- Tuple – An ordered set of values
- UML – Unified Modeling Language