UNIVERSITÉ FRANÇOIS - RABELAIS

DE TOURS

Année Universitaire : 2012-2013

HABILITATION À DIRIGER DES RECHERCHES

Leveraging query logs for user-centric OLAP

Discipline : Informatique

présentée et soutenue publiquement

par :
Patrick Marcel

le : 13/11/2012

JURY:
(Par ordre alphabtique)

Prénom	Nom	Grade	Établissement d'exercice
M. Mokrane	Bouzeghoub	Professeur des universités	Université de Versailles Saint-Quentin-en-Yvelines
M. Jérôme	Darmont	Professeur des universités	Université Lyon 2
M. Arnaud	Giacometti	Professeur des universités	Université François Rabelais Tours
M. Timos	Sellis	Professeur	National Technical University of Athens
M. Esteban	Zimanyi	Professeur	Université Libre de Bruxelles

Contents

List of Figures

List of Tables

List of Algorithms

Part I

Introduction

Chapter 1

The data deluge and its impact on OLAP users

This chapter introduces the context of the dissertation and outlines its contribution. Section 1.1 exposes the challenges around the management and querying of nowadays amounts of information for decision making purpose. Section 1.2 focuses on the data warehouse context, and points out the need for tools with better user-centric capabilities as a way to cope with the data deluge. Finally, Section 1.3 briefly introduces the contribution and presents the dissertation outline.

This chapter uses materials appearing in [71] as well as the description of the IT4BI (Information Technologies for Business Intelligence) Erasmus Mundus Master's Course[1], and especially the needs analysis underlying this programme.

1.1 The data deluge

In nowadays knowledge society, people and organizations are immersed in a constantly flowing torrent of information. According to a recent article of The Economist, mankind created 150 exabytes (billion gigabytes) of data in 2005. In 2010, it will create 1,200 exabytes [2]. Data continues to grow out of control. Since 2007, IDC has been sizing what it calls the Digital Universe, or the amount of digital information created and replicated in a year. Last year's report stated the following facts. *"In 2009, the Digital Universe grew by 62% to nearly 800,000 petabytes. In 2010, the Digital Universe will grow almost as fast to 1.2 million petabytes, or 1.2 zettabytes. This explosive growth means that by 2020, our Digital Universe will be 44 times as big as it was in 2009."*[3]

Businesses succeed or fail based largely on how effectively they collect, clean, transform, integrate, store, explore, analyze, and monitor this information to

[1] http://it4bi.univ-tours.fr
[2] http://www.economist.com/printedition/2010-02-27
[3] http://idcdocserv.com/925

predict future trends and make the best decisions. To succeed, organizations can no longer afford to treat information technologies as administrative tools, but need to embrace them as a strategic asset and embed them fully in their decision-making process. With such amounts of data, it is of paramount importance to be able to access relevant information efficiently, using sophisticated manipulation and search tools.

This is termed usually "big data" and asks extending traditional database architectures. Business Intelligence (BI) [26] promises an organization the capability of collecting and analyzing internal and external data to generate knowledge and value, providing decision support at the strategic, tactical, and operational levels. Since its inception 20 years ago, BI has become a huge industrial domain and a major economic driver, unaffected by the economic crisis, and is still growing fast, consistently mentioned among the top priorities of Chief Information Officers worldwide, as evidenced by analyst firms Forrester[4], IDC[5] and Gartner[6].

Business Intelligence has historically been based on a combination of Data warehousing [55, 44], the process of storing historical data in a structure designed for efficient processing, On-Line Analytical Processing (OLAP), the process of efficiently enabling common analytical operations on the multidimensional view of data, and Data mining [52], the mathematical and statistical methods necessary to transform this raw data into valuable information for making business decisions. More precisely, a data warehouse can be seen as a large database with a particular topology, where data is seen as a cube, shared by many analysts who have various interests and viewpoints, explored interactively by sequences of so-called OLAP queries [95] or from which knowledge is automatically extracted using data mining algorithms.

Data warehousing and OLAP are now mature technologies, having attracted a lot of attention from the academic and industrial community, and benefiting from the maturity of relational databases. Noticeably, the major part of this attention has been devoted to the efficient implementation and querying of the warehouse, and very little attention has been paid to the quality of the analysis and the ease of use of such technologies.

1.2 Are BI tools designed for BI users?

It has been recently observed that relational database management systems (RDBMSs) are quite uneasy to use, driving querying or navigation into huge amount of data a very tedious process, and that these systems should be more user-friendly [54]. BI tools, and especially OLAP tools, which are often implemented as Relational OLAP (ROLAP) engines and are thus based on relational database technology, also suffer from the same limitation. Recent studies showed that BI tools, although powerful, are often underused by decision makers, who

[4] http://www.forrester.com/rb/Research/market_overview_business_intelligence_software_market/q/id/55034/t/2

[5] http://www.idc.com/research/viewdocsynopsis.jsp?containerId=220987

[6] http://www.gartner.com/it/page.jsp?id=856714

frequently rely on static dashboards [86, 32, 17] instead of using the full potential of the tools. OLAP tools that go beyond simple reporting are considered tedious of use. Moreover, decision making is inherently a collaborative activity, which is largely overlooked by BI tools [17], as are the individual needs of the decision makers [90].

To answer these limitations, it is now commonly agreed that BI should benefit from a combination with Web 2.0 approaches (a focus on user empowerment, social networks, and community collaboration), a trend often referred to as BI 2.0 [105]. Indeed, in domains related to the Web like Information Retrieval or E-commerce, user-centric approaches like personalization or recommendation have been proved successful (see e.g., [27]). Such approaches are very relevant in a BI context, where the user may not accept to spend too much time conceiving the query to browse or analyze a data warehouse. In addition, she may not accept that the query's answer shows too many or too few results. And, even if the size of the answer is acceptable, she may be relieved to see the system automatically suggesting queries that will display other answers of interest, especially if she is left with the task of navigating the database to analyze the data it contains, which is typical of an OLAP user navigating a data warehouse using OLAP queries [95] that may return large answers. In such a context, being able to personalize or recommend queries is seen as particularly relevant [90].

It turns out that user-centric approaches in databases, like query personalization and query recommendation, are indeed getting more and more attention. Using preferences to personalize queries has been investigated for the last ten years [102], and recommending queries [101, 23] is emerging as a promising way of supporting the user browsing large databases.

In databases, query personalization and recommendation can be seen as techniques for computing a query q' from another query q, using information about the user and the context, called profile in what follows. More precisely, they can be defined as follows:

- Query personalization: given a database query q and a profile, compute a query $q' \subseteq q$[7] that has an added value w.r.t. the profile.
- Query recommendation: given a database query q and a profile, compute a query q' such that, in general, neither $q' \subseteq q$ nor $q \subseteq q'$, that has an added value w.r.t. the profile. Note that computing a query q' that includes q usually corresponds to query relaxation (see e.g., [56, 75]).

For instance, a user query asking for the *Average income of female employees in all European countries for the past year*, could be personalized to focus on France and Germany only, if the user profile indicates that in the recent past, this user was especially interested in these countries. On the other hand, the same query could lead to recommend to the user the query *Average income of male employees in all European countries for the past year* if the user profile indicates that other users similar to that user often evaluated this latter query after the former one.

[7] In the classical sense of query inclusion, i.e., whatever the database instance, the answer to q' is always a subset of the answer to q.

Achieving user empowerment without asking for too much formulation effort, while remaining proactive, can be done using approaches based on the history, i.e., the traces left by the users during former activities. Actually, it has already been pointed out the necessity to come up with flexible, powerful means for analyzing the issued queries, and decompose, store and handle them in a dedicated subsystem in order to better support any decisional task with the knowledge captured in the analytical queries [57]. Evidences of this need are given in [59], where the authors carried out tests demonstrating that browsing through past SQL query sessions helped the users by speeding up query composition.

Strangely enough, although highly relevant to BI in general and OLAP in particular, user-centric approaches, especially the ones based on leveraging past user queries, have attracted relatively little attention from the BI/data warehouse community.

1.3 Leveraging OLAP query logs for user-centric OLAP

This dissertation is a contribution to the huge task of developing user-centric OLAP. It focuses on the use of former queries logged by an OLAP server to enhance subsequent analyses.

This dissertation is organized in three main parts: Part II show how a log of OLAP queries can be modeled and constructed, and Part III focuses on how the log can be manipulated. These two parts provide the basic framework for taking advantage of OLAP query logs. Finally Part IV presents various personalization and recommendation techniques for achieving user-centric OLAP by leveraging the prepared log.

More precisely, after reviewing the basics of user-centric techniques like personalization and recommendation, and their use in a database context (chapter 2), Chapter 3 begins Part II by introducing the models of OLAP data, queries, sessions as sequences of queries and logs as sets of sessions. The corner stone of this modeling task is the modeling of queries, since the way queries are considered (fully, partially, or non evaluated) impacts how the log can be exploited for user-centric techniques. Given that not all query logs produced by an OLAP system respect such a model, Chapter 4 presents a technique to identify OLAP sessions, based on the semantic connections between queries, in a log where sessions need to be identified.

Part III begins with Chapter 5 that introduces a language for manipulating and querying query logs, based on the relational algebra. In particular, the operators of this language are parametrized by relations over sessions and queries. Some such relations are introduced in this chapter, and Chapter 6 is devoted to similarity measures to compare log objects (queries and sessions) that can be used to build such relations. In this chapter, various similarity measures are proposed, tailored for OLAP queries, and extending classical similarity measures for sequences.

The last main part presents various personalization and recommendation techniques taking advantage of query logs to support OLAP analyses. Part IV indeed begins with Chapter 7 that presents various techniques for extracting relevant knowledge from the query log. This knowledge includes simple preferences, navigational habits and discoveries made during former explorations. Chapter 8 describes approaches for personalizing a user's current OLAP queries, based on this user's former sessions. As for classical databases, these approaches can use dedicated operators for expressing preferences, or be based on query expansion. Chapter 9 presents approaches for computing collaborative query recommendations based on a multi-user log. These approaches can be based on information extracted from the current state of the database and the query, or be history based, i.e., leveraging the query log.

Finally chapter 10 concludes this dissertation by reviewing and discussing the contributions, and introduces future research directions.

Chapter 2

An overview of user-centric approaches in databases

This chapter provides an introduction to two popular user-centric approaches adopted by the database community, namely personalization and recommendation. More precisely, it will try to answer the following two questions:

- Given a database query q, how to to cope with too many or too few results? Personalization can be used to answer this question. If the query result is too large then being able to add preferences to this query gives a way of filtering out irrelevant results, or ranking the query results to focus on the most relevant first. Note that, personalization may also be used if the query result is too small, since then selection predicates, also called hard constraints, could be turned into preferences (or soft constraints) to weaken this query.
- Given a sequence of queries over a database, how to suggest queries to pursue the session? In this case, what the user did in the past, or alternatively, what similar users did in the past, can serve as a basis for recommending relevant queries to complement the current query answer.

Section 2.1 recall basic definitions on preference formulation and recommender systems. Section 2.2 introduces the criteria used throughout this dissertation to characterize the various approaches. Sections 2.3 and 2.4 respectively overview personalization and recommendation in databases. Finally, Section 2.5 lists the peculiarities of data warehouses and OLAP, that must be taken in to account for user-centric approaches.

This chapter uses materials appearing in [71].

2.1 Basics of preference expression and recommendation

2.1.1 Basics of preference expression

We begin by explaining the basics of preference modeling. A more comprehensive introduction can be found in [102].

Qualitative and quantitative preferences

Two types of approaches are used to express preferences. Qualitative approaches express relative preferences i.e., "I like a better than b". Such a preference is noted $a > b$ where $>$ is usually a Strict Partial Order (SPO). An SPO is a binary relation $>$ over a set O which is

- Irreflexive, i.e., for all $a \in O$, $\neg(a > a)$
- Asymmetric, i.e., for all $a, b \in O$, if $(a \neq b)$ and $(a > b)$ then $\neg(b > a)$
- Transitive, i.e., for all $a, b, c \in O$, if $(a > b)$ and $(b > c)$ then $(a > c)$

Given a preference relation $>$, the indifference relation $\sim$ is defined by: $(a \sim b)$ if $\neg(a > b)$ and $\neg(b > a)$. It expresses that a and b are not comparable. Particular partial orders of interest are Total Orders (TO) and Weak Orders (WO). A relation $>$ is a TO if for every a and b, either $(a > b)$ or $(b > a)$. $>$ is a WO if $>$ is a SPO and $\sim$ is transitive.

Example 2.1.1 *Consider the following database instance:*

Movies	*Author*	*Genre*	*Price*	*Duration*
t1	*Coen*	*Comedy*	*5*	*90*
t2	*Coen*	*Comedy*	*6*	*100*
t3	*Coen*	*Comedy*	*7*	*80*
t4	*Allen*	*Drama*	*7*	*120*
t5	*Lynch*	*Drama*	*5*	*150*

The preference "I prefer Lynch movies over Allen movies and Allen movies over Coen movies" entails that tuple t5 is preferred to tuple t4 and tuple t4 is preferred to tuples t1, t2, t3. As preferences are assumed to be transitive, we say that t5 dominates all the other tuples. Note that this preference says nothing e.g., for t1 and t2, neither for t1 and t3.

Quantitative approaches express absolute preferences, i.e., I (do not) like a to a specific degree. They are based on Scoring / Utility Functions. I like a better than b is defined by $u(a) > u(b)$, where u is a scoring function.

Quantitative approaches are often said to be less general than qualitative approaches in the sense that, in order to be representable using scoring functions, a preference relation has to be a WO, which implies that the corresponding indifference relation has to be transitive. But on the other hand, only scoring functions can accurately express the intensity of preferences.

Example 2.1.2 *The preference "I prefer Lynch movies over Allen movies and Allen movies over Coen movies" can be expressed by the following scoring function (assuming that scores range from 0 to 1):*

- *"I like Lynch" corresponds to a score of 0.9*
- *"I like Allen" corresponds to a score of 0.8*
- *"I like Coen" corresponds to a score of 0.5*

It can easily be seen that for instance, preference "I prefer cheaper movies, given that author and genre are the same" (i.e., t1 is preferred to both t2 and t3, but it is not preferred to t4 or t5) cannot be expressed with scoring functions.

Note that preferences can be expressed over sets of objects using preferences expressed over objects [21]. Given a set of objects O and a preference relation $>_O$ over O, subsets of O can be ordered w.r.t. $>_O$. An example of order over 2^O is given by: let X and Y be two subsets of O, X is preferred to Y, denoted $X >_S Y$, if for every $y \in Y$, there exists $x \in X$ such that $x >_O y$. If $>_O$ is a PO over O, then $>_S$ is a PO over 2^O.

Preference composition

Preferences can be defined extensionally under the form of relation instances, or intentionally. In the latter case, the intentional definition is called a model of preference. Preference composition follow the approach used to express preferences and thus can be qualitative or quantitative.

Qualitative composition In what follows, let T be a set of tuples and $>_1$ and $>_2$ be two preference relations over T. With single dimensional composition, preferences are expressed over a single relation. Common single dimensional composition includes Boolean Composition, Prioritized Composition, Pareto Composition and Lexicographic Composition.

Boolean composition involves a boolean operator, for instance:

- Intersection, i.e., $R = (>_1 \cap >_2)$ with $(t\ R\ t')$ if $(t >_1 t')$ and $(t >_2 t')$
- Union, i.e., $R = (>_1 \cup >_2)$ with $(t\ R\ t')$ if $(t >_1 t')$ or $(t >_2 t')$

Prioritized Composition imposes a priority of a preference over another. It is formally defined by $R = (>_1 \lhd >_2)$ with $(t\ R\ t')$ if $(t >_1 t')$ or $(\neg(t' >_1 t)$ and $(t >_2 t'))$

Pareto Composition assumes two preferences to be equally important. It is formally defined by[1]: $R = (>_1 \otimes >_2)$ with $(t\ R\ t')$ if $((t >_1 t')$ and $(t >_2 t'$ or $t \sim_2 t'))$ or $((t >_2 t')$ and $(t >_1 t'$ or $t \sim_1 t'))$.

[1]Note that alternative definitions using different semantics exist, as for instance the one given in [61].

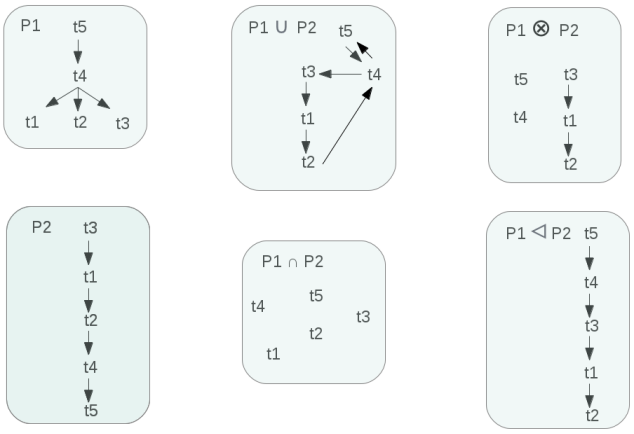

Figure 2.1: Examples of preference composition

Example 2.1.3 *Consider the two preferences: "I prefer Lynch movies over Allen movies and Allen movies over Coen movies", called P*1 *and "I also prefer shorter movies" called P*2*. Various compositions are illustrated by Figure 2.1.1. Composing them using intersection results in a particular SPO with no domination, that can be interpreted as every tuple is preferred since all tuples are un-dominated. Composing them with union violates irreflexivity and asymmetry, and thus the resulting relation is usually considered as not being a preference relation. Indeed, in this case, the resulting relation would mean for instance that both t5 should be preferred to t4 (according to P*1*) and t4 should be preferred to t5 (according to P*2*). Composing them with prioritization results in a total order reflecting P*1 *first and then P*2 *only when P*2 *does not contradict P*1*. More precisely, we have t5 preferred to t4 preferred to t3 preferred to t1 preferred to t2. Finally, composing them using Pareto results in a preference relation where only t3 dominates t1 and t1 dominates t2, and neither t4 dominates t5 nor t5 dominates t4 since P*1 *and P*2 *do not agree for these two tuples.*

Note that composition can be defined over $T \times T'$, if T and T' are two relations such that $>_1$ is a preference relation over T and $>_2$ is a preference relation over T'. For instance, lexicographic composition allows to define a lexicographic order over $T \times T'$. It is defined, for $t_1, t_1' \in T$ and $t_2, t_2' \in T'$, by: $R = (>_1 \odot >_2)$ with $(t_1, t_2)R(t_1, t_2)$ if $(t_1 >_1 t_1)$ or $(t_1 \sim_1 t_1)$ and $(t_2 >_2 t_2)$.

Pareto Composition can also be defined for multidimensional composition by, given $t_1, t_1' \in T$ and $t_2, t_2' \in T'$: $R = (>_1 \otimes >_2)$ with $((t_1, t_2)\ R\ (t_1', t_2'))$ if $((t_1 >_1 t_1')$ and $(t_2 >_2 t_2'$ or $t_2 \sim_2 t_2'))$ or $((t_2 >_2 t_2')$ and $(t_1 >_1 t_1'$ or $t_1 \sim_1 t_1'))$.

Note that preservation of properties (irreflexivity, etc.) under various kind of composition operators has been deeply studied (see [102] for more details).

Quantitative composition Quantitative composition is generally achieved through dedicated functions, like weighted sums, min, max, etc. An example of quantitative composition is, given preferences $P1$ modeled with $score_{P1}$ and preference $P2$ modeled with $score_{P2}$: $Score_{f(P1,P2)}(ti) = x \times score_{P1}(ti) + (1 - x) \times score_{P2}(ti)$ where x is some weight [64].

Example 2.1.4 *Consider the following preferences: "I prefer Lynch movies over Allen movies and Allen movies over Coen movies" (P1) expressed by:*

- *"I like Lynch" with* $score_{P1} = 0.9$
- *"I like Allen" with* $score_{P1} = 0.8$
- *"I like Coen" with* $score_{P1} = 0.5$

and "I also prefer shorter movies" (P2) expressed by:

- *"I like (duration=80)" with* $score_{P2} = 1$
- *"I like (duration=90)" with* $score_{P2} = 0.9$
- *"I like (duration=150)" with* $score_{P2} = 0.6$

Suppose we use the scoring function defined above to compose P1 and P2, with $x = 0.5$*. Then, for instance, the score of tuple* $t1 = \langle Coen, Comedy, 5, 90\rangle$ *would be:* $0.5 \times 0.5 + 0.5 \times 0.9 = 0.7$

2.1.2 Basics of recommender systems

We now briefly introduce the basics of recommender systems (see [3] for a more substantial presentation).

A recommender system is typically modeled as follows. Let I be a set of items (e.g., products in a typical e-commerce application) and U be a set of users (e.g., customers in a typical e-commerce application). Let f be an utility function with signature $U \times I \rightarrow R$ for some totally ordered set R (typically reals between 0 and 1). Recommending s' to u is to choose for the user u the item s' that maximizes the user's utility, i.e., $s' = argmax_I f(u, i)$. The function f can be represented as a matrix $M = U \times I$, that records for any user u in U, any item i in I, the utility of i for u, that is $f(u, i)$. The problem of recommending items to users is that this matrix is both very large and very sparse. Thus, many methods have been proposed for estimating the missing ratings. Moreover, achieving relevant recommendations, tailored for a particular user, is particularly difficult since it has been observed that everyone is a bit eccentric [41], meaning that recommending popular item is likely to be seen as not sufficient from the user's point of view.

In general, recommendation methods are categorized [3] into: (i) Content-based methods, that recommend items to the user u that are similar to previous items highly rated by u, (ii) Collaborative methods, that consider users similar (i.e. having similar profiles) to the one for which recommendations are to be computed as a basis for estimating missing ratings and (iii) Hybrid methods,

that combine content-based and collaborative ones. Note that [4] propose a multidimensional generalisation of this basic two-dimensional formulation, especially to support profiling and contextualisation.

Content-based recommendation

Typical content-based recommendation is based on the comparison between item profiles and user profiles. For instance, it can rely on the following principle:

1. Build item profiles by using selected features and providing a score for each feature.
2. Build user profiles from highly rated item profiles, typically by computing a weighted average of item profiles.
3. Compare user profiles with non-rated item profiles to estimate the missing ratings. Typical similarity measures include vector-based similarities like cosine.
4. Recommend to the user those non-rated items achieving the best similarity scores.

Example 2.1.5 *Consider the following matrix recording ratings:*

	Donuts	*Duff*	*Apple*	*Tofu*	*Water*	*Bud*	*Ribs*
Homer	*0.9*	*0.8*				*0.7*	
Marge			*0.8*		*0.6*		
Bart	*0.7*	*0.6*	*0.1*				*0.8*
Lisa	*0.2*			*0.8*	*0.6*		
Maggie	*0.6*			*0.5*	*0.6*		

Suppose that the features chosen for the profiles are: (contains sugar, ok for a diet). Item profiles are modelled as vectors recording a score for these two features. Suppose here that the scores are automatically computed from the items' nutrition facts. For instance, the profile for Tofu is $(0, 0.9)$*, the profile for Apple* $(0.4, 0.6)$*, the profile for Water is* $(0, 0.7)$*, and the profile for Ribs is* $(0.8, 0.1)$*. Then user profiles are also modeled as vectors recording scores for the two features, derived from the known ratings. For instance, Homer profile would be:* $(0.9 \times (0.9, 0) + 0.8 \times (0.6, 0.1) + 0.7 \times (0.6, 0.1))/3 = (0.8, 0.1)$[2]*. Lisa profile would be:* $(0.3, 0.8)$*. The similarity score for the user profile with the item profile estimates the missing ratings. For instance, the similarity score for Homer and Tofu is* $cosine((0.8, 0.1), (0, 0.9)) = 0.1$ *and that for Homer and Ribs is* $cosine((0.8, 0.1), (0.8, 0.1)) = 1$*. It is easy to see that Ribs will be recommended to Homer.*

The limitations of content-based approaches are the following: First finding a good set of features must be done very carefully since it impacts directly the

[2]This profile can be interpreted as: Homer contains sugar and is not ok for a diet.

score estimates and hence the quality of the recommendation. Another problem is that recommendations stick to the user profile. For instance, using the example above, Homer will never be recommended Tofu. Finally, this approach suffers from the cold-start problem, i.e., how to build a profile for a new user for whom no ratings are known.

Collaborative recommendation

The main idea of collaborative approaches is to benefit from all users' ratings when estimating a user's missing ratings. Depending on how the matrix is used (row-wise or column-wise), two techniques are distinguished, that are based on computing similarities among users or items:

- User-user collaborative approaches estimate the ratings for items based on ratings of similar users.
- Item-item collaborative approaches estimate the ratings for items based on ratings for all similar items (whatever the user).

Example 2.1.6 *To illustrate the user-user approach, suppose that all users are modeled as vectors having as many components as there exists items, the value of the component being the rating for the item, or 0 if the rating is not known*[3]*. For instance, Homer would be modeled as the following vector: (0.9,0.8,0,0,0,0.7,0). Similarity is computed between users, with cosine for instance, to find the users who are the most similar to the one for which the ratings are to be estimated. These users' ratings are derived to estimate the user's missing ratings. For instance, suppose that Bart is found the most similar to Homer, with a similarity score of 0.8. Then, given that Bart has a score for Ribs and Homer has not, Bart's score is used to estimate Homer's, by weighting Bart's score with the similarity between Bart and Homer, i.e.,* 0.8×0.8 *is our example.*

The limitations of the collaborative approach are that it relies on heavy precomputation, and that new users or new items, for which no ratings are known, cannot be taken into account.

Hybrid methods

Hybrid approaches are used to overcome the limitations of both previous approaches. Such approaches include the aggregation of a content-based score with a collaborative score, the addition of content-based to collaborative filtering, or the use of item profiles to cope with the new item problem.

2.2 Categorisation of the approaches

In this section, we present the criteria used to describe and categorize the approaches presented in the next sections and chapters. In what follows, the

[3]Note that it means treating unknown ratings and low ratings similarly.

term profile is used to denote the information associated with the user and the context, that are leveraged by the approach. This profile may include user preferences, description of past activities of the database users (like e.g., the database query log), as well as external information.

We first adopt the criteria introduced in [43], that are mostly used to differentiate personalization approaches.

- Formulation effort asked to the user: some approaches require the user to manually specify profile elements for each query (high formulation effort), while in others they are inferred from the context and the user's past actions (low formulation effort).
- Prescriptiveness: some approaches use profile elements as hard constraints that are added to a query (prescriptive approaches) while others use them as soft ones: tuples that satisfy as much profile criteria as possible are returned even if no tuples satisfies all of them (non prescriptive approaches).
- Proactiveness: some approaches suggest new queries based on the navigation log and on the context (proactive approaches), while others change the current query or post process its results before returning them to the user (non proactive approaches).
- Expressiveness: some approaches allow only the formulation of basic personalization criteria (low expressiveness) while others allow more complex formulation (high expressiveness).

To precisely distinguish between the type of data needed in the profile, especially for recommendation techniques, we also use the taxonomy proposed in [101]. There, three categories are identified:

- Current-state approaches, exploiting the content and schema of the current query result and database instance. Current-state approaches can be based either on (i) the local analysis of the properties of the result of the posed query or (ii) the global analysis of the properties of the database. In both cases systems exploit (i) the content and/or (ii) the schema of the query result or the database.
- History-based approaches, using the query logs.
- External sources approaches, i.e., approaches exploiting resources external to the database, like ontologies, web pages, etc.

Note that these categories are not exclusive in the sense that an approach can be both current-state and history-based.

We illustrate these criteria with the following example: consider an approach where both the query and the profile are inferred from the past users activities on the database, the profile being composed of complex preferences and being used to rank the query results. Such an approach would correspond to the following category: low formulation effort, proactive, non prescriptive, high expressiveness, history-based.

2.3 An overview of personalization in databases

Query personalization in databases include two types of approaches:

- The use of explicit preference operators in queries to specify profile elements. The most representative works in this category include Winnow [28], Preference SQL [60] and Skyline [20]. This type of approaches requires high formulation effort, is not prescriptive, not proactive, but is highly expressive.
- The rewriting (expansion) of regular database queries based on a profile. The most representative work is initiated in [64]. This approach requires a low formulation effort, is prescriptive, not proactive and has low expressiveness.

Note that in these two categories, the profile is only given in terms of the constants, attributes and tuples of the database. Thus, using the criteria introduced above, all the approaches are current-state approaches.

2.3.1 Use of a dedicated operator

The basic definition of the operator computing dominating tuples is the following. Given a relation r with schema $sch(r)$ and a preference C over $sch(r)$ defining a preference relation $>_C$, the Winnow operator [28], denoted w_C, is defined by: $w_C(r) = \{t \in r | (\nexists t' \in r)(t' >_C t)\}$.

This operator can be used to order the query answer. Indeed, the answer to a query q can be partitioned according to C, i.e., $q = w_C(q) \cup w_C(q - w_C(q)) \cup \ldots$ meaning that the answer can be presented by displaying $w_C(q)$ first, then $w_C(q - w_C(q))$, etc.

Example 2.3.1 *Consider the examples given in Section 2.1.1. Suppose that preference C is "I prefer drama". The query "What are my most preferred affordable movies?" can be expressed by: $w_C(\sigma_{Price<7}(Movies))$. The answer can be presented by displaying t5 first, and then t1 and t2.*

This operator has been used with various syntaxes for expressing models of preferences, of which we give two illustrations [20, 60].

The work of Kiessling [60] enrich the SQL syntax with a PREFERRING clause that enables the user to specify in each query the model of preference, through the use of of specific preference constructors. Each constructor can be used to express a specific atomic preference. Preferences can be composed with Pareto or prioritization.

Example 2.3.2 *Consider the following queries expressed with Preference SQL:*

1. SELECT * FROM Movies PREFERRING HIGHEST(Duration)
2. SELECT * FROM Movies PREFERRING Genre IN (Drama,Thriller)

The model of preference specified by the first query is the following: for some duration x and y, $(x >_{HIGHEST} y)$ if value x is greater than value y, meaning that movies with highest durations will be preferred. For the second query, the model of preference says that Drama and Thriller movies will be preferred over any other genre. More formally, if x and y are two movie genres, $(x >_{IN(Drama,Thriller)} y)$ if $x \in \{Drama, Thriller\}$ and $y \notin \{Drama, Thriller\}$.

A restricted form of Kiessling's SQL extension is the addition of the Skyline operator to SQL [20]. This is a restriction in the sense that, originally, Skyline queries feature only numerical attributes and Pareto composition. This operator is defined as follows. The syntax of a skyline clause is:

SKYLINE OF d_1 MIN, ..., d_k MIN
d_{k+1} MAX, ..., d_l MAX
d_{l+1} DIFF, ..., d_m DIFF

The semantics of such a clause is that a tuple $p = (p_1, \ldots, p_n)$ dominates a tuple $q = (q_1, \ldots, q_n)$ if:

$p_i \leq q_i$, for $i = 1, \ldots, k$

$p_i \geq q_i$, for $i = k+1, \ldots, l$

$p_i = q_i$, for $i = l+1, \ldots, m$

2.3.2 Query expansion

In the absence of a dedicated preference operator, a regular user query can be processed and transformed with preferences, resulting in another regular query that is typically a subquery of the initial one. We use the work of [64] to illustrate this approach.

In this work, preferences come from a user profile which is modeled as a graph of atomic quantitative preferences of the form *(selection condition, score)*, where *selection condition* is a regular selection predicate (that may be used to join two tables) and *score* is a real between 0 and 1 indicating the intensity of the preference. Atomic preferences are composed using a very simple principle: composition of preference $(s1, v1)$ with preference $(s2, v2)$ results in preference $(s1 \wedge s2, v1 \times v2)$.

Query expansion is performed as follows. First, given a query, the k most relevant preferences are selected from the profile. Then the selected preferences are added as hard constraints to the query, and the query is finally executed.

Example 2.3.3 *Consider the following user query:* SELECT title FROM Movies WHERE duration < 120. *Suppose that the best preference selected from the profile of the user who wrote the query is "I like Lynch as Author". Then the query is modified, resulting in:* SELECT title FROM Movies WHERE duration < 120 AND Author='Lynch'. *Note that in this example, if the query is evaluated over the instance given in Example 2.1.1, then the result is empty.*

This work has been extended to take into account constraints like result cardinality or execution time [65].

2.4 An overview of recommendations in databases

Recently, to our knowledge, there has been only two attempts to formalize database query recommendations for exploration purpose [24, 101]. All the approaches presented below require a low formulation effort, are proactive (a query is suggested) and prescriptive (no soft constraint are added to the suggested query) and offer a low expressiveness. Note that a recent work [4] proposed an SQL-like language to let the user formulate the way recommendations are computed, for reducing prescriptiveness, but requiring a higher formulation effort. Following [101], we use the categories introduced Section 2.2 to categorize these approaches with respect to the type of information composing the profile.

2.4.1 Current state

In [101], the authors focus on the current state approach and propose two techniques to recommend queries based on the database instance and/or the current query answer. The first technique, called local analysis, analyzes the answer to the user query to discover patterns and uses these patterns to recommend. The second technique, called global analysis, extends this principle to the entire database instance. The instance would have to be analyzed off-line, for example to discover correlations among attribute values.

Example 2.4.1 *Consider the current query:* SELECT Author, Genre FROM Movies WHERE Duration > 100. *Suppose that by analyzing this query answer, it is found that the result has a lot of tuples whose genre is Drama. Then, a possible recommendation would be:* SELECT Author, Genre FROM Movies WHERE Genre='Drama'. *Suppose now that a global analysis of the database instance shows that value "Coen" for Author is correlated with value "Comedy" for Genre. Then, if the current query is:* SELECT * FROM Movies WHERE Author='Coen' *a recommendation would be:* SELECT * FROM Movies WHERE Genre='Comedy'.

2.4.2 History based

For [24], the problem of query recommendation is viewed as a sessions $\times$ tuples matrix. With this approach, a query is represented as a vector whose arity is the number of tuples of the database instance. The basic approach considers that each component of such a query is either a 1, if the query used the tuple, or a 0 otherwise. A session is also represented by a binary vector which is the logical OR of the vectors of the queries of the session. Consider a particular session S_c called the current session. Recommendations for S_c are computed as follows. First, sessions similar to S_c are found, using some vector similarity measures like e.g., cosine. For those sessions closest to S_c, the queries they contain are also compared to S_c using the same similarity measure. Finally, the queries of those sessions that are the most similar to S_c are recommended.

In subsequent works [23], the authors focus on fragments (attributes, tables, joins and predicates) of queries and consider thus a sessions × query fragments matrix. In this work, the matrix is used column-wise in the sense that recommendation computation relies on fragment similarity instead of session similarity.

We conclude this section by noting that recommendation also makes sense to assist the user writing a query. For instance, the SnipSuggest approach [58] is a collaborative approach that uses a query log to provide on-the-fly assistance to users writing complex queries. The query log is analyzed to construct a graph whose nodes are the fragments appearing in queries and edges indicate the precedence. The edges are labelled with the probability that a fragment follows another fragment in the logged queries. Given the beginning of a current query, the graph is used to complete the query with the fragment that is the most likely to appear.

Example 2.4.2 *Suppose that the query log contains only the following two queries:* SELECT Title, Genre FROM Movies WHERE Actor="C. Lee" *and* SELECT Title FROM Movies WHERE Author="Allen". *Suppose that a user starts writing a query with only* SELECT. *It can be then suggested the attribute* Title *since this attribute is the most likely to appear according to the query log.*

2.5 Conclusion: Requirements for user-centric approaches in data warehouses

This chapter presented an overview of personalization and recommendation approaches in databases, that transform queries with respect to a user profile. We also introduced criteria to describe them: how the profile is expressed, used and how complex it is, and what type of data is used for the profile (external to the database, or internal, such as the instance, the query log, the schema).

To conclude this chapter, we introduce the requirements for user-centric approaches in the context of OLAP analysis of data warehouses. As evidenced by e.g., [25, 96, 89] basic peculiarities of typical data warehouses can be summarized by:

1. A data warehouse is a read-mostly database and its instance has an inflationist evolution (data are added, never or very seldom deleted). It is for instance likely that a user issues periodically *similar sequences of queries* more than once, in the sense that *queries may not be fully identical.*

2. A data warehouse is a database shared by *multiple users, mostly executives, whose interests are diverse and may vary over time.* It is argued in [16, 89, 45, 90] that *user preferences are of particular importance in data warehouse exploration.* It would for instance be important to *issue recommendations computed from other users' habits* (e.g., in a collaborative filtering fashion) and at the same time *respecting the user interests.*

3. A data warehouse has a particular schema that reflects a known topology, often called the lattice of cuboids, which is systematically used for navigation [52]. *Roll-up and drill-down operations that allow to see facts at various levels of detail are very popular in this context.*

4. A typical analysis session over a data warehouse is a sequence of queries having an analytical goal, each one written based on the past results of the session. They may be expressed in a *dedicated query language (like e.g., MDX[4])*, may produce *large results that are usually visualized as cross-tabs.* Moreover, *the session has a sense w.r.t. some expectations.* For instance, the user may assume a uniform distribution of the data [94, 95] or that two populations follow the same distribution [85]. Sessions (as sequences of queries) are of particular importance in this context since *with this sequence of queries, the user navigates to discover valuable insights w.r.t. her expectations or assumptions.*

The following parts of this dissertation aim at developing various user-centric approaches responding to these requirements, with a focus on the use of the query log. In particular, the next chapter introduces models for OLAP query logs, that will serve as bases for such approaches.

[4] http://msdn.microsoft.com/en-us/library/ms145506.aspx

Part II

Modeling and constructing query logs

Chapter 3

Log modeling

This chapter provides the basic formal setting used throughout this dissertation. It first describes the multidimensional framework starting with data (Section 3.1) and then various models for queries, sessions and logs (Section 3.2).

This chapter relies on materials published in [16, 37, 39, 12, 92, 9].

3.1 Multidimensional data and query languages

In this section, we recall the classical definitions and terminology used in data warehousing and OLAP. Basic knowledge is assumed on the relational model and query languages, as can be found in e.g., [2]. We simply recall that an attribute A has a domain $dom(A)$, and, given a relation instance I over a schema including A, the active domain of an attribute A denoted $adom(A)$ corresponds to the values in $\pi_A(I)$. $t(i)$ denotes the i^{th} value of a tuple t.

In this dissertation, we consider **facts**, as subjects of analysis, placed in the n-dimensional space produced by the analysis dimensions. A **dimension** contains an aggregation hierarchy of **levels** representing different granularities (or levels of detail) to study data. A **fact** contains analysis indicators known as **measures** (which, in turn, can be regarded as fact attributes). A level of detail for each dimension produces a **group by set** (also called **base**) in which place the measures.

3.1.1 Hierarchies and dimensions

To keep the formalism simple, we consider cubes under a ROLAP perspective, described by a star schema [62]. More precisely, we consider that a dimension consists of one hierarchy, and we consider simple hierarchies without branches, i.e., consisting of chains of levels.

Definition 3.1.1 (Levels and members) *Let $\mathcal{L}$ be a set of attributes called levels, and for $L \in \mathcal{L}$, a member is an element of $Dom(L)$.*

Roll-up and Drill-down are two partial mappings from $\mathcal{L}$ to $\mathcal{L}$ defined by: Given two levels L_j and L_k, $Rollup(L_j) = L_k$ if there exists a functional dependency $L_j \rightarrow L_k$ or undefined otherwise, and $Drilldown(L_j) = L_l$ if there exists a functional dependency $L_l \rightarrow L_j$ or undefined otherwise.

Definition 3.1.2 (Hierarchy) *A hierarchy h_i is a set $Lev(h_i) = \{L_0, \ldots, L_d\}$ of levels together with a* roll-up *total order $\succeq_{h_i}$ of $Lev(h_i)$, which is such that, for any L_j and L_k in $Lev(h_i)$, $L_k \succeq_{h_i} L_j$ if $Rollup(L_j) = L_k$.*

For each hierarchy h_i, the bottom level L_0 of the order determines the finest aggregation level for the hierarchy. Conversely, the top level L_d has a single possible value and determines the coarsest aggregation level.

A dimension is a relation used to represent a hierarchy.

Definition 3.1.3 (Dimension) *A dimension D for a hierarchy h_i is a relation with schema $sch(D) = Lev(h_i) = \{L_0, \ldots, L_d\}$, such that L_0 is the primary key of D. A dimension table for D is an instance of D.*

The set of members in a dimension D is denoted $\pi^*(D) = \bigcup_{i=0}^{d} \pi_{L_i}(D)$. Given two levels L_j, L_k of a dimension D such that $L_k = Rollup(L_j)$, we use $m_k \succeq m_j$ to denote that $\langle m_j, m_k \rangle \in \pi_{L_j, L_k}(D)$. $\succeq$ is a transitive relation. Given a dimension D for a hierarchy h_i, we note m_i^{All} the coarsest member of the hierarchy, i.e., the member such that $\nexists m \in \pi^*(D)$ with $m \succeq m_i^{All}$.

3.1.2 Multidimensional schemata, group-by sets and references

Definition 3.1.4 (Multidimensional Schema) *A* multidimensional schema *(or, briefly, a* schema*) is a triple $\mathcal{M} = \langle A, H, M \rangle$ where:*

- *A is a finite set of* levels*, whose domains are assumed pairwise disjoint,*
- *$H = \{h_1, \ldots, h_n\}$ is a finite set of* hierarchies*, (such that the $Lev(h_i)$'s for $i \in \{1, \ldots, n\}$ define a partition of A);*
- *a finite set of* measure *attributes M, each defined on a numerical domain $Dom(m)$.*

A group-by set includes one level for each hierarchy, and defines a possible way to aggregate data. It is sometimes referred to as a base in what follows. A reference (or coordinate) of a group-by set is a point in the n-dimensional space defined by the levels in that group-by set.

Definition 3.1.5 (Group-by Set and reference) *Given a schema $\mathcal{M} = \langle A, H, M \rangle$, let $Dom(H) = Lev(h_1) \times \ldots \times Lev(h_n)$; each $G \in Dom(H)$ is called a* group-by set *of $\mathcal{M}$. Let $G = \langle a_{k_1}, \ldots, a_{k_n} \rangle$ and $Dom(G) = Dom(a_{k_1}) \times \ldots \times Dom(a_{k_n})$; each $g \in Dom(G)$ is called a* reference *(or a coordinate) of G.*

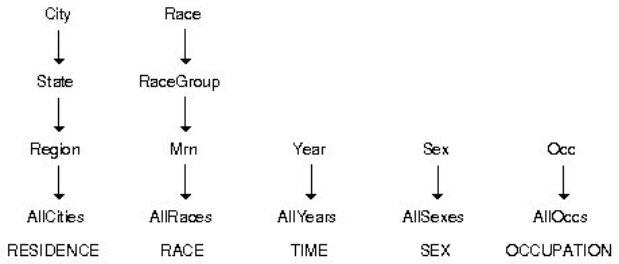

Figure 3.1: Roll-up orders for the five hierarchies in the CENSUS schema (Mrn stands for MajorRacesNumber)

Let $\succeq_H$ denote the product order[1] of the roll-up orders of the hierarchies in H. Then, $(Dom(H), \succeq_H)$ is a lattice, that we will call *group-by lattice*, whose top and bottom elements are $G^\perp = \langle DIM_1, \ldots, DIM_n \rangle$ and $G^\top = \langle ALL_1, \ldots, ALL_n \rangle$, respectively.

Example 3.1.1 *We introduce a running example. IPUMS is a public database storing census microdata for social and economic research [74]. Its* CENSUS *multidimensional schema includes the five hierarchies whose roll-up orders are shown in Figure 3.1, and measures* AvgIncome, AvgCostGas, AvgCostWatr, *and* AvgCostElect.

More formally, its schema is $\mathcal{CENSUS} = \langle A_{CENSUS}, H_{CENSUS}, M_{CENSUS} \rangle$ *with:*

$A_{CENSUS} = \{City, State, Region, AllCities, Race, RaceGroup, Mrn, AllRaces, Year, AllYears, Sex, AllSexes, Occ, AllOccs\}$,

$H_{CENSUS} = \{\mathsf{RESIDENCE}, \mathsf{RACE}, \mathsf{TIME}, \mathsf{SEX}, \mathsf{OCCUPATION}\}$,

and $M_{CENSUS} = \{\mathsf{AvgIncome}, \mathsf{AvgCostGas}, \mathsf{AvgCostWatr}, \mathsf{AvgCostElect}\}$.

For instance, hierarchy RESIDENCE *is the set of levels* $\{City, State, Region, AllCities\}$ *with* $AllCities \succeq_{\mathsf{RESIDENCE}} Region \succeq_{\mathsf{RESIDENCE}} State \succeq_{\mathsf{RESIDENCE}} City$.

Examples of group-by sets are:

$$G_0 = G^\perp = \langle \mathsf{City}, \mathsf{Race}, \mathsf{Year}, \mathsf{Occ}, \mathsf{Sex} \rangle$$
$$G_1 = \langle \mathsf{Region}, \mathsf{Mrn}, \mathsf{Year}, \mathsf{Occ}, \mathsf{Sex} \rangle$$
$$G_2 = G^\top = \langle \mathsf{AllCities}, \mathsf{AllRaces}, \mathsf{AllYears}, \mathsf{AllOccs}, \mathsf{AllSexes} \rangle$$

Specialization relation over references Given two references r and r', we consider the classical relation over references defined by: $r \succeq r'$ if, for all dimensions D_i with hierarchy $\succeq_i$, either $r(i) = r'(i)$ or $r(i) \succeq r'(i)$.

[1]The product order of n total orders is a partial order on the Cartesian product of the n totally ordered sets, such that $\langle x_1, \ldots, x_n \rangle \succeq \langle y_1, \ldots, y_n \rangle$ iff $x_i \succeq y_i$ for $i = 1, \ldots, n$.

3.1.3 Facts and cubes

A schema is populated with facts, each recording a useful information for the decision-making process. A fact is characterized by a group-by set G that defines its aggregation level, by a reference of G, and by a value for each measure.

Definition 3.1.6 (Fact) *Given a schema $\mathcal{M} = \langle A, H, M\rangle$, a group-by set $G \in Dom(H)$, and a measure set $V = \{v_1, \ldots, v_m\} \subseteq M$, a fact is a couple $f_{G,V} = \langle g, v\rangle$, where $g \in Dom(G)$ and $v \in Dom(V)$, where $Dom(V) = Dom(v_1) \times \ldots \times Dom(v_m)$. The space of all facts for $\mathcal{M}$ is*

$$\mathcal{F}_{\mathcal{M}} = \bigcup_{G \in Dom(H), V \subseteq M} (Dom(G) \times Dom(V))$$

Example 3.1.2 *An example of fact is $f_{G_1,\mathsf{AvgIncome}} = \langle\langle$'Pacific', 'White', '2008', 'Dentist', 'Male'$\rangle, \langle 600\rangle\rangle$. Its reference is $\langle$'Pacific', 'White', '2008', 'Dentist', 'Male'$\rangle$.*

An instance of a schema is a set of facts $F \subseteq \mathcal{F}_{\mathcal{M}}$, such that no two facts characterized by the same coordinate and measure, exist in F. A Fact table F is a relation representing facts by tuples.

Definition 3.1.7 (Fact table) *Given a schema $\mathcal{M} = \langle A, H, M\rangle$, and a measure set $V = \{v_1, \ldots, v_m\} \subseteq M$, a fact table F is a relation with schema $\{L_1^0, \ldots, L_n^0, v_1, \ldots, v_m\}$ where for all $i \in [1, n]$, L_i^0 is the finest aggregation level of h_i, i.e., the primary key of some dimension table D_i. $L_1^0, \ldots, L_n^0$ is the primary key of F.*

Finally, an n-dimensional cube is defined as the classical $n + 1$ relation instances of a star schema.

Definition 3.1.8 (Cube) *An n-dimensional cube $C = \langle D_1, ..., D_n, F\rangle$ over a multidimensional schema $\mathcal{M} = \langle A, H, M\rangle$ is a set of relation instances where $D_1, \ldots, D_n$ are dimension tables and F is a fact table. The set of dimensions of a cube $C = \langle D_1, ..., D_n, F\rangle$ is noted $\mathcal{D}(C) = \{D_1, ..., D_n\}$.*

Definition 3.1.9 (Cells) *Let $C = \langle D_1, ..., D_n, F\rangle$ be a cube, and let $L_i \in sch(D_i)$, for all $i \in [1, n]$. A cell c is a tuple $c = \langle m_1, \ldots, m_n, x_1, \ldots, x_m\rangle$ where $\langle m_1, \ldots, m_n\rangle$ is a reference over $L_1, \ldots, L_n$ and $\langle x_1, \ldots, x_m\rangle \in dom(V)$.*

Given a cube C and aggregation functions $agg_1, \ldots, agg_m$, a cell whose reference is $\langle m_1, \ldots, m_n\rangle$ over $L_1, \ldots, L_n$, is the result of the relational query (where $\overset{\circ}{\bowtie}$ is the outerjoin):

$$\{\langle m_1, \ldots, m_n\rangle\} \quad \overset{\circ}{\bowtie} \quad \pi_{v_1,\ldots,v_m}(\sigma_{L_1=m_1 \wedge \ldots \wedge L_n=m_n}(\pi_{L_1,\ldots,L_n;agg_1(v_1)\to v_1,\ldots,agg_m(v_m)\to v_m}(F \bowtie D_1 \bowtie \ldots \bowtie D_n)))$$

The x_i are called the measures of the cell. In what follows we will use $measures(c)$ to denote the measures of the cell c. $ref(C)$ denotes the set of all possible references of a cube $C = \langle D_1, ..., D_n, F\rangle$, i.e., $ref(C) = \times_{i=1}^{N} \pi^*(D_i)$, $i \in [1, n]$.

Data cube Given m aggregation functions $agg_1, \ldots, agg_m$, the data cube [47] of a cube $C = \langle D_1, ..., D_n, F \rangle$, denoted $CubeBy(C)$, is a set of relation instances $C' = \langle D_1, ..., D_n, F' \rangle$ where:

$$F' = \cup_{D_1.L_1^{k_1} \in sch(D_1), ..., D_N.L_N^{k_N} \in sch(D_N)} [\rho_{D_1.L_1^{k_1} \to D_1, ..., D_N.L_N^{k_N} \to D_N} (\pi_{D_1.L_1^{k_1}, ..., D_N.L_N^{k_N}, agg_1(v_1), ..., agg_m(v_m)} (F \bowtie D_1 \bowtie \ldots \bowtie D_N))]$$

The specialization relation over references is extended to cells as follows: For two cells c, c' of an n-dimensional cube C, c is more general than c', noted $c \succeq_c c'$, if $r \succeq r'$ where r is the reference of c and r' is the reference of c'. Note that this relation corresponds to the one used in the cube lattice (see e.g., [66]).

Example 3.1.3 *A cube over the* CENSUS *schema consists of a fact table* census *with schema {city, race, year, sex, occupation, avgIncome, avgCostGas, avgCostElec, avgCostWatr}, and dimension tables* RESIDENCE, RACE, TIME, SEX, OCCUPATION. *with respective schema {city, region, state, allCities}, {race, raceGroup, MRN, allRaces}, {year, allYear}, {sex, allSexes}, {occ, allOccs}.*

3.1.4 Expressing multidimensional queries

Formally speaking, a query is a partial mapping from database instances to database instances [2], which is distinct from its expression in a given language. In this dissertation, we clearly distinguish between a query and a query expression, which is needed since a query can be specified by various query expressions. We design by query expression a specification written in a given query language. Out of this expression, a logical model of a query can be built, as will be seen in the next section.

Though multidimensional queries can often be expressed using the extended relational algebra [88], or a subset of it like *GPSJ (Generalized Projection / Selection / Join* [49]), a better characterization in a dedicated language is often preferred. The literature abounds with multidimensional languages, most of them coming from the academic world ([5, 50, 51, 107] to name a few), some from the industry like SQL99 and MDX. In this section we briefly present only two languages, MDA, a formal algebra presented in [91], which captures the cube-query in [62], and MDX, often referred to as the de-facto standard. Note that, for simplicity, we consider queries centered on a single schema.

The Multidimensional Algebra MDA

MDA was proven to be closed, complete (regarding the cube-query in [62]) and minimal, and consists of the following operators (we suggest to check Figure 3.2, where dots and triangles represent measures in a cell, for grasping their intuition). All the operators are unary except for drill-across and set operators, which operate over two cubes. We give below an informal description, details can be found in [1].

- **Selection** ($\sigma_p cube$): By means of a logic predicate p compound of clauses over levels, this operator allows to choose the subset of points of interest out of the whole n-dimensional space.

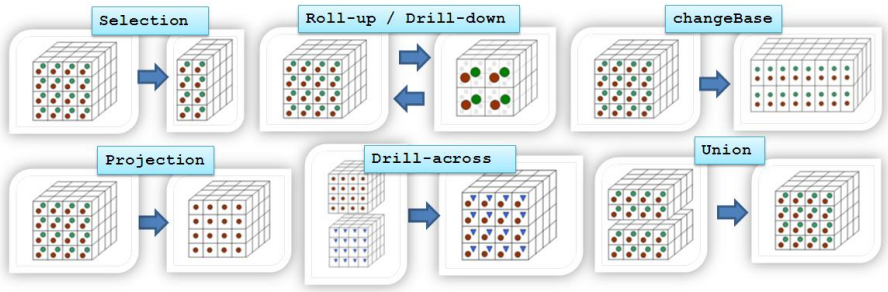

Figure 3.2: Conceptual exemplification of the MDA operators

- **Roll-up** ($\gamma^{level_i \rightarrow level_j}_{f(measure_1),\ldots,f(measure_n)} cube$): It groups data instances in the cube based on an aggregation hierarchy. This operator modifies the granularity of data by means of a many-to-one relationship which relates instances of two levels in the same hierarchy, corresponding to a part-whole relationship. About drill-down (i.e., the counterpart of roll-up, represented with the same formalization but with a one-to-many relationship between $level_i$ and $level_j$), it can only be applied if we previously performed a roll-up and did not loose the correspondences between instances. Therefore, a drill-down is often seen as the sequence of roll-up operations allowing to reach a desired granularity from the finest granularity.

- **Projection** ($\pi_{measure_1,\ldots,measure_n} cube$): It selects a subset of measures.

- **ChangeBase** ($\chi_{base_1 \rightarrow base_2} cube$): This operator 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 to replace the current base by one of the alternatives, if more than one set of dimensions identifying the data instances (i.e., alternative bases) exist.

- **Drill-across** ($cube_1 \bowtie cube_2$): This operator 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.

- **Set Operations** ($cube_1 \Theta cube_2$): These operators allow to operate two cubes if both are defined over the same n-dimensional space. We consider union ($\cup$), difference ($\setminus$) and intersection ($\cap$). Set operations are defined for cubes having the same schema and such that, in the case of union, whenever two cells have the same reference, they also have the same measures.

The expressive power of this algebra is thoroughly discussed in [1]. Briefly, it fully matches the well-known cube-query pattern presented in [62]. For this reason, it is assumed to be expressive enough for capturing analytical efforts.

MDX

The MDX (*MultiDimensional eXpressions*[2]) language is a de-facto standard for querying multidimensional databases. Some of its distinguishing features are the possibility of returning query results that contain data with different aggregation levels and the possibility of specifying how the results should be visually arranged into a multidimensional representation. Albeit very powerful in terms of expressiveness, it is not thoroughly formalized and is not closed under composition [70]. In this dissertation we consider MDX queries that aggregate data at one or more group-by sets, optionally select them using a predicate in CNF, and return one or more measures. The semantics of such an MDX query is that of a union of GPSJ queries[3] whose group-by sets are the cross product of n sets of levels, one for each hierarchy. This semantics corresponds to the following subset of MDX:

- Clauses **SELECT**, **FROM**, **WHERE** are supported.
- All functions for navigating hierarchies are supported: **AllMembers**, **Ancestor**, **Ascendants**, **Children**, etc.
- All functions for manipulating sets of members or tuples are supported (**Crossjoin**, **Except**, **Exists**, **Extract**, **Filter**, **Intersect**, etc.) except the union.
- All functions for manipulating members/tuples are supported.

Example 3.1.4 *We give two expressions of a query over the* CENSUS *schema, that asks for the Female average income by city and race. The MDX formulation is:*

SELECT Race.MEMBERS **ON COLUMNS,**
City.MEMBERS **ON ROWS**
FROM CENSUS
WHERE (Female,AvgIncome)

This query, expressed in MDA, is:
$\sigma_{Sex='Female'}(\gamma^{Occ\rightarrow allOccs}_{avg(avgIncome)}\ (\gamma^{years\rightarrow allYears}_{avg(avgIncome)}(\pi_{avgIncome}(CENSUS)))))$.

We have distinguished between a query and its expression in a given language. To leverage query logs, we need to derive information from this expression and other database objects, which we call a query model. This is the topic of the following section.

[2] http://msdn.microsoft.com/en-us/library/ms145506.aspx

[3] A GPSJ query takes form $\pi_{a_{k_1},\ldots,a_{k_n},Aggr}\sigma_p(\chi)$ where, in our context: χ is the star join between the fact table and the n dimension tables; p is a selection formula in CNF; $\{a_{k_1},\ldots,a_{k_n}\}$ is a group-by set; and $Aggr$ is a list of aggregations of the form $\alpha_j(m_j)$, where m_j is a measure and α_j is an aggregation operator.

3.2 Modeling queries and logs

As pointed out above, using a query model to reason about query allows to be independent from the query expression. In this section we first review the literature for works where database queries are modeled, and we give three examples of logical query models used in the subsequent chapters. We use the following definitions.

- Query: the function mapping a database instance to another database instance.
- Query expression: the expression of the query in a given query language.
- Query model: a logical representation of the query that can include various information about it, derived from its expression and potentially the database objects like the instance, the schema or even the query log.

Note that a query model may correspond to more than one query expressions, while a query expression corresponds to only one query model. Usually a function is used to construct the model from the query expression (and potentially the database objects). When the context is clear, we will use the term query to refer to the query expression or model.

3.2.1 Query models in the literature

In this subsection, we review how queries are logically modeled from query expression. We start by distinguishing the data structure used (vector, set, etc.) to represent the query. We see that two categories are very often used: vector and set (or set of sets). In the first category (vector), queries are modeled as a vector of some features with either a score or a binary value for each feature. In the second category (set), query are modeled either by one or more sets. In the latter case, one set is used for representing a particular part of the query, like e.g., the attributes of the query schema (projection or SELECT clause) or the table names in the cross product (FROM clause).

The information used to model the query can be taken from the query expression itself, e.g., under the form of its fragments (selection predicate, projection, etc.) and/or taken from the database over which the query is to be evaluated, e.g., the database instance, schema (including usable physical structures), etc.

The part of the query expression that can be used ranges from the simple uninterpreted query text ([110]) to the full list of query fragments [34]. When the fragments are used, all or only parts of them can be taken into account (e.g., for selection predicate, only the selection attribute is used in [6]).

On the other hand, the information used to model the query can also be taken from other sources related to the database queried. More precisely, that can be:

- The database instance: the query model can rely on the extension of

the database or not, e.g., if the query result or the active domain of the database attributes are used.

- The statistics used by the query optimizer, like e.g., table sizes, attribute cardinalities, etc.
- The database schema, e.g., information on key definition or which index can be used to process a selection.
- The query log, if the query model relies on the other queries that have previously been launched on the database, e.g., a query is modeled in terms of its link with other queries or how many times it appears in the log.

This is summarized in the table 3.1, where S, P and C denote respectively selection predicates, projections or group by sets, and cross product.

Ref.	*Model*	*Source*	*Part of fragment used*
[48]	sets	S, P, C	attributes, values, table names
[24]	vector	db instance, log	
[8]	vector	S, P, log	attributes, values
[6]	vector	S, db instance	attributes, values
[13]	vector	S, P, log	attributes
[35]	vector	S, C, db statistics	
[101] (1)	vector	log	
[101] (2)	set	db instance	
[37]	set	db instance	
[93]	sets	S, P	attributes
[42]	set	P, db schema, db statistics	attributes
[110]	string	SQL sentence	
[109]	graph	S, P, C	attributes, table names

Table 3.1: Query models

We now introduce three particular query models for multidimensional queries, used throughout this dissertation.

3.2.2 No evaluation: Queries as a collection of fragments

A multidimensional query can be modeled as a set of fragments, extracted from the query expression [9].

Definition 3.2.1 (Query Fragments from a multidimensional schema) *Given schema $\mathcal{M} = \langle A, H, M \rangle$, a* query fragment *is either a level in A, a measure in M, or a simple Boolean predicate involving a level and/or a measure. A* qf-set *is a set of query fragments.*

A multidimensional query over a schema $\mathcal{M} = \langle A, H, M \rangle$ can be modeled by a qf-set that includes at least one level for each hierarchy in H and at least one measure in M. For instance, representing an MDX query as a qf-set q means:

1. Including a fragment m in q for each measure m returned by the MDX query.
2. Including a fragment a in q for each level a used in the MDX query to aggregate data.

3. Including a fragment ($a \in V$) in q for each simple predicate on a level or measure a used in the MDX query to filter data.

We give an example of fragments extracted from MDX queries [9].

Example 3.2.1 *The MDX query on the* CENSUS *schema*

```
SELECT AvgIncome ON COLUMNS,
      Crossjoin(OCCUPATION.members,
            Crossjoin(Descendants(RACE.AllRaces,RACE.Mrn),
            Descendants(RESIDENCE.AllCities,RESIDENCE.Region))) ON ROWS
FROM CENSUS WHERE TIME.Year.[2009]
```

is the union of four GPSJ queries:

$$\pi_{\mathsf{AllCities,AllRaces,Occ,Year,AllSexes},AVG(\mathsf{AvgIncome})}\sigma_{\mathsf{Year}=2009}(\chi_{\mathsf{CENSUS}})$$
$$\pi_{\mathsf{AllCities,Mrn,Occ,Year,AllSexes},AVG(\mathsf{AvgIncome})}\sigma_{\mathsf{Year}=2009}(\chi_{\mathsf{CENSUS}})$$
$$\pi_{\mathsf{Region,AllRaces,Occ,Year,AllSexes},AVG(\mathsf{AvgIncome})}\sigma_{\mathsf{Year}=2009}(\chi_{\mathsf{CENSUS}})$$
$$\pi_{\mathsf{Region,Mrn,Occ,Year,AllSexes},AVG(\mathsf{AvgIncome})}\sigma_{\mathsf{Year}=2009}(\chi_{\mathsf{CENSUS}})$$

and is represented by the qf-set $q = \{\mathsf{Region}, \mathsf{AllCities}, \mathsf{Mrn}, \mathsf{AllRaces}, \mathsf{Occ}, \mathsf{Year}, \mathsf{AllSexes}, \mathsf{AvgIncome}, (\mathsf{Year} = 2009)\}$.

An alternate representation is to model the query by a triple better structuring the 3 components of a multidimensional query: A measure set, a set of selection predicates and a group-by set.

Definition 3.2.2 (Fragment-based OLAP query model) *The model of a* query *over schema* $\mathcal{M} = \langle L, H, M\rangle$ *is a triple* $q = \langle G, P, Meas\rangle$ *where:*

1. $G \in Dom(H)$ *is the query group-by set;*
2. $P = \{p_1, \ldots, p_n\}$ *is a set of Boolean predicates, one for each hierarchy, whose conjunction defines the* selection predicate *for q; they are of the form* $l = v$, *or* $l \in V$, *with* l *a level,* v *a value,* V *a set of values. Conventionally, we note* $p_i = TRUE_i$ *if no selection on* h_i *is made in* q *(all values being selected);*
3. $Meas \subseteq M$ *is the measure set whose values are returned by* q.

Example 3.2.2 *An example of model of a query on the* CENSUS *schema is:*

$$q_1 = \langle G^{\perp}, TRUE, \mathsf{AvgIncome}\rangle$$

where $G^{\perp} = \langle \mathsf{AllCities}, \mathsf{AllRaces}, \mathsf{AllYears}, \mathsf{AllOccs}, \mathsf{AllSexes}\rangle$ *and* $TRUE = \{TRUE_{\mathsf{RESIDENCE}}, TRUE_{\mathsf{RACE}}, TRUE_{\mathsf{TIME}}, TRUE_{\mathsf{OCCUPATION}}, TRUE_{\mathsf{SEX}}\}$.

3.2.3 Partial evaluation: Queries as sets of references

A query can be modeled as a set of tuples, obtained with a partial evaluation of the query expression. For instance, a multidimensional query over a star schema can be evaluated only over the dimension tables, resulting in a set of references [16, 37, 12]. Note that obtaining this set of references from a query expression can be computed efficiently when dimension tables fit in main memory. In that case, a query can be expressed intentionally as a tuple of sets, one set of members in each dimension. The cross-product of these sets is a set of references, which forms the query model.

Definition 3.2.3 (Query modeled as a set of references) *Given an n-dimensional cube $C = \langle D_1, ..., D_n, F \rangle$ over a multidimensional schema $\mathcal{M} = \langle A, H, M \rangle$, let R_i be a set of members of dimension $D_i, \forall i \in [1, n]$. A query expression $q = \langle R_1, ..., R_n \rangle$ is a tuple of sets of members, one for each dimension D_i of C. Given such an expression, the query model of q is the set of references $R_1 \times ... \times R_n$.*

Example 3.2.3 *consider the following MDX query on the* CENSUS *schema:*

```
SELECT AvgIncome ON COLUMNS,
       Crossjoin(SEX.members,{RESIDENCE.Region.Pacific,
       RESIDENCE.Region.Atlantic}) ON ROWS
FROM CENSUS WHERE TIME.Year.[2001]
```

An example of model of this is:

$$\{Pacific, Atlantic\} \times \{AllRaces\} \times \{2001\} \times \{AllOccs\} \times \{Male, Female\}$$

3.2.4 Full evaluation: Queries as their results

Finally, a query can be modeled as the set of its answer when evaluated over a database instance [39]. It can be defined as follows: Given a cube instance and a query expression, the query models is the set of facts extracted from the data cube by the query expression. Obviously, this set can be empty.

3.2.5 Modeling sessions and logs

In this dissertation, we use the term log to refer to the actual analytical queries launched over a data warehouse. The term workload, very common in the literature on query optimization, is usually seen as a set of queries (or more precisely query expressions). This dissertation develops the idea that the actual query log is more than a set of queries, and can be seen as a more complex structure of which we give a more accurate definition.

Indeed, in an OLAP context, queries are often not isolated from one another, as shown by [93], that reports a study of a query log of 18 users over a two months period in a large chemical company. The log showed that users interactively formulate their next query based on the result of the previous query, illustrating

the navigational nature of an analytical session (or analysis) over a data cube. This is especially the case in discovery driven analysis [94, 95, 97]. We thus define an analytical session, or session for short, as a sequence of query expressions.

Definition 3.2.4 (OLAP Session) *Let $C = \langle D_1, ..., D_n, F\rangle$ be an n-dimensional cube over a multidimensional schema $\mathcal{M} = \langle A, H, M\rangle$, and S_C be a set of query expressions over C in a given language. A session s of k query expressions $s = \langle q_1, \ldots, q_k\rangle$ over C is a function from an ordered set $pos(s)$ of integers (called positions) of size k to S_C.*

As query models are constructed from query expressions, a session can also be seen as a sequence of query models.

A log L is a finite set of sessions, noted $L = \{s_1, \ldots, s_p\}$.

Definition 3.2.5 (Log) *Let $C = \langle D_1, ..., D_n, F\rangle$ be an n-dimensional cube over a multidimensional schema $\mathcal{M} = \langle A, H, M\rangle$. A log L is a finite set of sessions over C.*

Note that if sessions are assumed to be launched one after the other (i.e., the sessions over the data warehouse are not concurrent), then a log itself can be seen as a sequence of query expressions, hence as a session.

We denote the set of query expressions of a session s by $queries(s)$ and the set of query expressions of a log L by $queries(L)$. We note $q \in L$ for a log L if $q \in queries(L)$.

Example 3.2.4 *An example of OLAP session of length 3 on the* CENSUS *schema is $s = \langle q_1, q_2, q_3\rangle$, where the fragment-based query models are:*

$$q_1 = \langle G^{\perp}, TRUE, \{\mathsf{AvgIncome}\}\rangle$$
$$q_2 = \langle G_1, TRUE, \{\mathsf{AvgIncome}\}\rangle$$
$$q_3 = \langle G_1, \{\mathsf{Year} = 2011, TRUE_{RACE}, \ldots\}, \{\mathsf{AvgIncome}\}\rangle$$

where $TRUE =$
$\{TRUE_{\mathsf{RESIDENCE}}, TRUE_{\mathsf{RACE}}, TRUE_{\mathsf{TIME}}, TRUE_{\mathsf{OCCUPATION}}, TRUE_{\mathsf{SEX}}\}$.

Note that the user here applied a roll-up operator to move from q_1 to q_2, and a slice operator to move from q_2 to q_3.

3.3 Conclusion

This chapter introduced various models for OLAP query logs. Logs are modeled as sets of sessions, sessions being modeled as sequences of OLAP queries. Three main ways are given for modeling queries: as unevaluated collections of fragments (group by sets, sets of selection predicates, sets of measures), as sets of references obtained by partially evaluating the query over dimensions, and as their answers. Importantly, answers or sets of references can be obtained from the collection of fragments.

Intuitively, these three models can be seen as three different focuses on how the user interacts with the cube, where the importance is either the way the query is written (unevaluated queries), or the part of the cube queried (partially evaluated queries), or the values retrieved (fully evaluated queries). It is also worth noting that both the effectiveness and efficiency of the user-centric approach depends on the query model used. Indeed, a trade-off exists, that can be expressed as follows. On the one hand, modeling queries as their results allows to leverage the knowledge extracted with the query, but requires important resources to store or recompute the query answer, especially in a data warehousing context. On the other hand, using only information derived from the query expression may give rise to efficient approaches, but not knowing the data extracted with the query may prevent valuable deductions, like two different query expressions lead to identical results on a given instance.

Obviously, not all query logs produced by an OLAP system, especially those based on the ROLAP technology, naturally respect this model. The next chapter provides a technique for obtaining a log respecting this model, from a sequence of relational queries.

Chapter 4

Constructing the log

As noted in the previous chapter, the query log obtained from an OLAP server or DBMS may not respect the model of log introduced in Definition 3.2.5. In particular, as it is often the case in a ROLAP setting, OLAP queries can be expressed under the form of SQL (or relational) expressions. The log may simply be a long sequence of such expressions that may not be organized as a set of sessions. And even though the queries are organized as sessions, they may not be logically connected toward the exploration of an analytical goal. Consequently, OLAP semantics must be extracted from each expression, under the form of OLAP operations, and sessions must be identified.

This chapter introduces a technique to obtain a log respecting the model defined in the previous chapter, i.e., a set of sessions of OLAP queries. Section 4.1 starts with an informal description of the approach, and presents the running example. In Section 4.2, we show how to characterize each query in the query log by means of the *MultiDimensional Algebraic* (MDA) operators presented in chapter 3. Then, Section 4.3 explains how to normalize this characterization so it can be compared to other characterizations. Finally, Section 4.4 indicates how to detect that two consecutive queries are logically connected, which enables to detect sessions.

This chapter relies on material published in [92] and [106]. Note that in this chapter, the term query refers to query expression.

4.1 Principle and running example

In this section, we outline the principle of the technique and introduce a running example to illustrate it.

4.1.1 Principle

The technique relies on the following three steps:

1. We start by characterizing each analytical relational query by means of MDA, which gives multidimensional sense to the query. This characteri-

zation is an expression in the MDA language that is called a *MAC* (for Multidimensional Algebraic Characterization).

2. We define a normal form for queries expressed in MDA, and propose a set of *equivalence rules*, based on those of the relational algebra to transform MDA expressions. We show how to use these rules to pull the MDA operators up the algebraic structure, and produce a *Normalized MAC*, that will be called an *NMAC*.

3. We consider that two queries q_1, q_2 could be coalesced in the same session if we only need to add a few MD operators to obtain q_2's output from q_1's NMAC. In other words, *bridging* is the process of producing the same result as q_2 by adding operators to q_1's NMAC (thus, we are *bridging* from q_1 to q_2). Based on the length of the bridge found, we can decide whether it makes sense or not to consider both queries in the same session.

4.1.2 A running example

We now introduce the relational queries that will be used in the subsequent examples of this chapter. We consider a scenario inspired by the TPC-DS benchmark [104, 88], where a relational database is accessed with queries expressed in SQL. The database schema consists of the following relations (where foreign keys are represented as $attr_1$ ($\rightarrow attr_2$)):

```
catalog_sales (cs_date (→ date), cs_store (→ store), cs_customer (→ customer),
cs_product (→ product), cs_quantity, cs_amount),
date_dim (date, month, quarter, year),
store_dim (store, address, city, state, region),
customer_dim (customer, name, address, city, state, profession, branch),
product_dim (product, description, line)
```

Q1	Q2	Q3
select state, sum(cs_quantity) from catalog_sales, date_dim, store_dim where cs_product = '1' and cs_date= date and cs_store = store group by year, state	select month, state, sum(cs_quantity) from catalog_sales, date_dim, store_dim where cs_product = '1' and cs_date = date and cs_store = store group month, state	select month, state, sum(cs_quantity) from catalog_sales, date_dim, store_dim where cs_product = '1' and region = 'SE' and cs_date = date and cs_store = store group by month, state

Figure 4.1: Exemplification of three SQL analytical queries within the same session

Consider queries in Figure 4.1, extracted from the query log. q_1 asks for the total sales by state and year for a product, q_2 disaggregates sales by month and q_3 focuses on the south-east region. It turns out that these queries can be expressed in MDA. Query q_1 can be bridged with query q_2. Indeed, it can be detected that it corresponds to a drill-down from the year level to the month level. Furthermore, q_2 can be in turn bridged with q_3 since it corresponds to adding a selection over region. The three queries thus, should be characterized as a single session.

We will also use the more complex query q_4, which unites two cubes with the total sales by customer state and month, for year 1999, for both the south-east and south-west regions:

```
SELECT month, state, SUM(cs_quantity) AS sales
FROM catalog_sales, date_dim, customer_dim
WHERE cs_date = date AND cs_customer = customer AND year = 1999
AND region = 'SE'
GROUP BY month, state
UNION
SELECT month, state, SUM(cs_quantity) AS sales
FROM catalog_sales, date_dim, customer_dim
WHERE cs_date = date AND cs_customer = customer AND year = 1999
AND region = 'SW'
GROUP BY month, state;
```

In what follows, we consider that these queries will be applied over a cube Sales whose schema is $\langle A_{Sales}, H_{Sales}, M_{Sales}\rangle$ with:
$A_{Sales} = \{date, month, quarter, year, store, city, state, region, customer, profession, branch, product, line\}$,
$H_{Sales} = \{$

- $\langle\{date, month, quarter, year, allYears\}, allYears \succeq year \succeq quarter \succeq month \succeq date\rangle$,
- $\langle\{store, city, state, region, allRegions\}, allRegions \succeq region \succeq state \succeq city \succeq store\rangle$,
- $\langle\{customer, profession, branch, allBranches\}, allBranches \succeq branch \succeq profession \succeq customer\rangle$,
- $\langle\{customer, c\_city, c\_state, allStates\}, allStates \succeq c\_state \succeq c\_city \succeq customer\rangle$,
- $\langle\{product, line, allLines\}, allLines \succeq line \succeq product\rangle\}$

and $M_{Sales} = \{quantity, amount\}$.

4.2 Multidimensional characterization of queries

We now present how to translate relational expressions into MDA expressions.

4.2.1 Mapping OLAP operators with relational operators

In [91], it is shown how the MDA operators presented in Chapter 3 can be expressed in terms of restricted operators of the relational algebra. We take advantage of this work to identify the MDA operators, given a relational query. First, we briefly refresh the relationship between both algebras and later, we discuss how to formulate the MAC of a relational query. Without loss of generality, we denote by raw data (over which apply the MDA operators) the universal relationship of the tables in the cross product of a relational query, i.e., the join between the fact table and all the dimension tables in the context of a star schema.

Operators	σ	π	$\bowtie$	$\cup$	"Group by"	"Aggregation"
Selection	✓					
Projection		$\checkmark_{Measures}$				
Roll-up					✓	$\checkmark_{Measures}$
Drill-across		✓	✓			
ChangeBase			✓			
Union				✓		

Table 4.1: Mapping between the relational and MDA operators

Table 4.1 synthesizes the mapping between MDA operators and extended relational operators. In the table, a ✓ indicates that there is a translation from the relational operator into the MDA one. $\checkmark_{measures}$ indicates that the MDA operator is equivalent to the relational one but it can be only applied over measures. If the translation of a MDA operator combines more than one relational operator, they appear ticked in the same row. We address the reader to [91] for a detailed justification of this table.

Note that this table can be used to unambiguously translate relational operators into their MDA counterpart. Nevertheless, translation is possible only if the constraints inherent to the multidimensional model are satisfied.

Definition 4.2.1 (Constraints inherent to the multidimensional model) *These constraints are:*

1. *Fact/Dimension dichotomy must be preserved, which is reflected in that members and measures are disjoint.*
2. *Summarizability necessary conditions (as in [67]) 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***: $cube_1 \rightleftharpoons cube_2$ must be one-to-one.*

These constraints can be easily verified using the schema of the database. A relational query that could not be fully formulated in terms of MDA operators would not make multidimensional sense and thus should not be considered as an OLAP query.

4.2.2 From relational queries to multidimensional queries

We now introduce the grammar that is used to derive the MDA expression, that is, the MAC of the relational query. By definition, a MAC is a tree-shaped structure. Like in the relational algebra, this is because of binary operators. 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, i.e., correspond to the same group by set, before being able to collapse them with a binary operation. 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*).

Finally, we talk about the *pivotal node* (i.e., the first binary ancestor of the root) as the tree node dividing the MAC into two well-differentiated layers: the *structural layer* and the *presentation layer*. 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).

Definition 4.2.2 (Grammar used for the MAC) *The grammar capturing the semantics of a MAC is as follows (χ, σ, π, γ, $\cup$, $\bowtie$ represent the MDA operators).*

$\mathcal{MAC} \rightarrow rawData\ \mathcal{NP} \mid (\mathcal{MAC} \cup \mathcal{MAC})\mathcal{NP} \mid (\mathcal{MAC} \bowtie \mathcal{MAC})NP \qquad \mathcal{Q} \rightarrow \mathcal{CB}\ \mathcal{S}\ \mathcal{R}\ \mathcal{P}$
$\mathcal{NP} \rightarrow \mathcal{Q} \mid \mathcal{Q}\ \mathcal{NP} \quad \mathcal{CB} \rightarrow \emptyset \mid \chi\mathcal{CB} \quad \mathcal{S} \rightarrow \emptyset \mid \sigma\mathcal{S} \quad \mathcal{R} \rightarrow \emptyset \mid \gamma\mathcal{R} \quad \mathcal{P} \rightarrow \emptyset \mid \pi\mathcal{P}$

The process used to translate a relational expression q_r into its MAC q_m is summarized by Algorithm 1.

Algorithm 1 Obtaining the MAC of a relational query

Input: q_r: A query expressed in the relational algebra
Output: The MAC of q_r if it exists, or an exception
1: Parse q_r using the grammar of Definition 4.2.2 and Table 4.1
2: **if** at least one constraint of Definition 4.2.1 is not satisfied **then**
3: **return** with exception "no multidimensional characterization"
4: **else**
5: **return** q_m as the result of the parsing

We illustrate this algorithm on the running example.

Example 4.2.1 *The MAC of q_1 is (where raw data is the universal relation for* `catalog_sales` $\times$ `store_dim` $\times$ `date_dim`*, i.e., tables in the FROM):*

$$\gamma^{date \rightarrow year}_{sum(cs\_quantity)}(\quad \gamma^{store \rightarrow state}_{sum(cs\_quantity)}(\quad \gamma^{product \rightarrow allLines}_{sum(cs\_quantity)}(\quad \gamma^{customer \rightarrow allBranches}_{sum(cs\_quantity)}(\quad \gamma^{customer \rightarrow allSstates}_{sum(cs\_quantity)}(\quad \pi_{cs\_quantity}(\quad \sigma_{cs\_product='1'}(raw\_data)))))))$$

For the sake of readability, roll-up operations that aggregate to the coarsest granularity level in some dimension are omitted in subsequent examples. Thus

the MAC of q_1 is simply noted:

$$\gamma^{date\rightarrow year}_{sum(cs\_quantity)}(\gamma^{store\rightarrow state}_{sum(cs\_quantity)}(\pi_{cs\_quantity}(\sigma_{cs\_product='1'}(raw\_data))))$$

The MAC of query q_1 expresses the roll-up to the state and year levels, the projection over the measure (cs_quantity) and the selection (of the product with code '1').

The MAC of query q_2 is:

$$\pi_{cs\_quantity}(\gamma^{date\rightarrow month}_{sum(cs\_quantity)}(\gamma^{store\rightarrow state}_{sum(cs\_quantity)}(\sigma_{cs\_product='1'}(raw\_data))))$$

The MAC of q_4 this query is:

$$(\sigma_{region='SE'}(\gamma^{date\rightarrow month}_{sum(cs\_quantity)}(\gamma^{customer\rightarrow state}_{sum(cs\_quantity)}(\pi_{cs\_quantity}(\sigma_{year=1999}(raw\_data))))))$$

$$\cup$$

$$(\sigma_{region='SW'}(\gamma^{date\rightarrow month}_{sum(cs\_quantity)}(\gamma^{customer\rightarrow state}_{sum(cs\_quantity)}(\pi_{cs\_quantity}(\sigma_{year=1999}(raw\_data))))))$$

Note that this MAC contains two NP, namely:

$$\sigma_{region='SE'}(\gamma^{date\rightarrow month}_{sum(cs\_quantity)}(\gamma^{customer\rightarrow state}_{sum(cs\_quantity)}(\pi_{cs\_quantity}(\sigma_{year=1999}(raw\_data)))))$$

and

$$\sigma_{region='SW'}(\gamma^{date\rightarrow month}_{sum(cs\_quantity)}(\gamma^{customer\rightarrow state}_{sum(cs\_quantity)}(\pi_{cs\_quantity}(\sigma_{year=1999}(raw\_data))))).$$

In this MAC, the pivotal node is the union, because in this case it represents the structural part. There is no presentation layer in this case, since the union is also the root.

4.3 Normalization of multidimensional expressions

In this section, we explain how the MAC of the relational expression is turned into a normal form. We first start by the characterization of this normal form and then present an algorithm to obtain it.

4.3.1 A normal form for multidimensional expressions

Intuitively, 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, MACs can contain more than one NP (some of them interleaved in the structural layer for aligning binary operators), although only the root-most NP (i.e., the one amid the pivotal and root nodes) represents the presentation layer. Thus, in the normalized MAC, operators are placed in the presentation layer (i.e., to the MAC root-side) whenever possible. If an operator remains stuck in the structural part after normalization then, it is needed for retrieving tuples rather than for presentation purposes. We precise this in the following definition.

Definition 4.3.1 *A normalized MAC (NMAC) is a MAC with the following properties:*

i) It is composed of a presentation layer, that is a sequence of unary operations at the root of the tree, and of a structural layer, a tree of binary and unary operations whose root is a binary operation connecting the structural layer and the presentation layer.
ii) All operations that are not needed for aligning the inputs of binary operators appear in the presentation layer. NPs stuck in the structural part are needed for aligning the inputs of binary operators (and not for presentation purposes).
iii) The minimum number of Q (see Definition 4.2.2) appear at each NP, each potentially containing χ, σ and γ, in this order.
iv) π can only appear in the topmost Q of every NP, and following the order imposed by the containment of attributes.

Furthermore, we force a partial order aimed at facilitating the NPs comparison, as follows:

i) γ in Q will be sorted by dimension and then aggregation level.
ii) σ in Q will follow the inverse order the user posed them.
iii) χ in Q will follow the inverse order the user posed them.

Example 4.3.1 *Consider the MAC of query q_4 given in the previous example. This MAC is not in normal form, since properties (ii) (many operations can be pulled up through the union), (iii) (operations in the NPs are not sorted properly) and (iv) (the projections are not in the topmost position) do not hold.*

A NMAC is produced by using the equivalence rules described below.

4.3.2 Equivalence rules for the multidimensional algebra

In our approach we benefit from the algebraic structure proposed, and we use a set of equivalence rules to pull the MD operators up the algebraic structure.

Operator	Projection	Roll-up	Selection	ChangeBase
Projection	×	✓	✓	✓
Roll-up	✓	∼	∼	∼
Selection	✓	✓	✓	∼
ChangeBase	✓	∼	∼	✓
Drill-across	✓	↗↖	↗↖	↗↖
Union	↗↖	↗↖	↗↖	↗↖

Table 4.2: MDA equivalence rules

Thus, the MDA equivalence rules (shown in Table 4.2) are an immediate consequence of considering the MDA operator semantics over the relational algebra equivalence rules and considering the constraints introduced in Section 4.2. The meaning of each cell in the table is the following: if the MDA operator in the column *can be pulled up*[1] the operator in the row, the cell is ticked

[1]We recall that a MAC is a tree-shaped structure and consequently, we talk about *pulling up an operator* through the structure.

("✓"). If there is a conflict, the cell is crossed ("×"). Like in the relational algebra equivalence rules, a "∼" denotes a partial conflict: the operator can be pulled up whenever the row operator does not remove the attribute needed by the column operator. For example, a selection can only be pulled up a roll-up if the attribute used to select is not rolled-up. Finally, a "↗↖" refers to binary operators. A unary operator can be pulled up the binary operator if it appears in both branches as explained below. For example, we can only pull up a projection through a union if the same measures are projected in both branches.

4.3.3 Normalization algorithm

The normalization algorithm is just a postorder traversal of the MAC, considering that the nodes to visit are NPs and binary operations (thus, being a postorder algorithm, for each binary operator, it first visits its branches and later the binary operator itself). 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 leafs-side), we pull it up in the direction of the root as much as possible within the NP, following the rules in the white and light gray cells of Table 4.2.
b) Next, for each binary operator we visit, if both left and right branches are non-empty NPs and some operation coincides in their topmost Qs which can be pulled up through its successors in Q according to the light and dark gray cells of Table 4.2, the unary operator is pulled up from both and added once at the leafs-side of the parent NP of the binary operator. Note that, every binary operator will always have a parent NP (in the trivial case, the one containing the root node). Only exception, according to Table 4.2, is that it is not necessary that a projection must coincide at both branches of a drill-across to be pulled up.

Example 4.3.2 *Consider again the MAC of query q_4. Following a postorder traversal, we would first visit both NPs,which would be sorted to result in:*

$$\pi_{cs\_quantity}(\gamma^{customer\rightarrow state}_{sum(cs\_quantity)}(\gamma^{date\rightarrow month}_{sum(cs\_quantity)}(\sigma_{year=1999}(\sigma_{region='SE'}(raw\_data)))))$$

and

$$\pi_{cs\_quantity}(\gamma^{customer\rightarrow state}_{sum(cs\_quantity)}(\gamma^{date\rightarrow month}_{sum(cs\_quantity)}(\sigma_{year=1999}(\sigma_{region='SW'}(raw\_data)))))$$

Afterwards, we would visit their parent, yielding the following:

$$\pi_{cs\_quantity}(\gamma^{customer\rightarrow state}_{sum(cs\_quantity)}(\gamma^{date\rightarrow month}_{sum(cs\_quantity)}(\sigma_{year=1999}(\sigma_{region='SE'}(raw\_data) \cup \sigma_{region='SW'}(raw\_data)))))$$

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

4.4 Detecting sessions

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, called 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 parts* coincide by comparing their *presentation layers*. Indeed, if structural parts do not coincide, data retrieved by the two queries are too much different to be part of the same session. From the presentation layer of a NMAC n, the following components can be extracted:

- The query group by set $gb(n)$, by looking at the γ operations.
- The query predicates $pred(n)$, by looking at the σ operations.
- The query measure set $meas(n)$, by looking at the final π operation.

NMACs whose structural part coincide can thus be characterize by means of the fragment based OLAP query model introduced in the previous chapter, i.e., for a NMAC n, the triple $components(n) = \langle gb(n), pred(n), meas(n)\rangle$.

Example 4.4.1 *Consider q_2's MAC, which is normalized:*

$$\pi_{cs\_quantity}(\gamma^{date\rightarrow month}_{sum(cs\_quantity)}(\gamma^{store\rightarrow state}_{sum(cs\_quantity)}(\sigma_{cs\_product='1'}(raw\_data))))$$

This query can be modeled as:

$$\begin{array}{l}\langle\langle month, state, allBranches, allStates, allLines\rangle,\\ \{product = 1, TRUE_{YEAR}, \ldots, TRUE_{STATE}\},\\ \{cs\_quantity\}\rangle\end{array}$$

Since MDA is close and every operator has its inverse, by definition, given two NMACs n_1 and n_2, we can transform $components(n_1)$ into $components(n_2)$ by means of a finite set of MDA operators applied to n_1. Furthermore, given MDA's minimality, we know which operators can be applied in order to do so: group-by sets can be changed with roll-up or drill-down, predicates can be changed with σ or $\cup$, and measure sets can be changed with π or $\bowtie$. Algorithm 2 bridges two NMACs, resulting in a set S of MDA operators.

The produced bridge is then evaluated to decide whether n_1 and n_2 are similar enough, for instance by evaluating $|S|$ against a threshold, and if so we consider both NMACs to belong to the same session. 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 to keep track of their logical connection.

Algorithm 2 Bridging two NMACs

Input: n_1, n_2: two NMACs
Output: A set of MDA operation, or an exception
Variables: S: A set of MDA operations

1: $S = \emptyset$
2: **if** the structural parts of n_1 and n_2 do not coincide **then**
3: **return** with exception "queries non bridgeable"
4: **else**
5: **for** each level $l_1 \in gb(n_1), l_2 \in gb(n_2)$ with $l_2 \succeq l_1$ **do**
6: $S = S \cup \{\gamma^{l_1 \to l_2}\}$ ▷ Roll-up to those coarser n_2 levels
7: **for** each level $l_1 \in gb(n_1), l_2 \in gb(n_2)$ with $l_1 \succeq l_2$ **do**
8: $S = S \cup \{\gamma^{l_1 \to l_2}\}$ ▷ Drill-down to those finer n_2 levels
9: **if** $\exists l_2 \in gb(n_2)$ such that $\forall l_1 \in gb(n_1), \neg(l_1 = l_2 \vee l_2 \succeq l_1 \vee l_1 \succeq l_2)$ **then**
10: $S = S \cup \{\chi_{gb(n_1) \to gb(n_2)}\}$ ▷ Base is to be changed
11: **if** $\exists s \in pred(n_1) \setminus pred(n_2)$ **then**
12: $S = S \cup \{\sigma_{\{s \in pred(n_2)\}}\}$ ▷ Select only with predicates of n_2
13: **if** $\exists s \in pred(n_2) \setminus pred(n_1)$ **then**
14: $S = S \cup \{\cup(raw\ data)\}$ ▷ values not in n_1 need to be fetched
15: **if** $\exists m \in meas(n_1) \setminus meas(n_2)$ **then**
16: $S = S \cup \{\pi_{\{m \in meas(n_2)\}}\}$ ▷ Measures not in n_2 are projected out
17: **if** $\exists m \in meas(n_2) \setminus meas(n_1)$ **then**
18: $S = S \cup \{\bowtie (raw\ data)\}$ ▷ Measures not in n_1 need to be fetched
19: **return** S

Example 4.4.2 *Consider q_2's MAC, which is normalized:*

$$\pi_{cs\_quantity}(\gamma^{date \to month}_{sum(cs\_quantity)}(\gamma^{store \to state}_{sum(cs\_quantity)}(\sigma_{cs\_product='1'}(raw\_data))))$$

This query can be bridged with q_1. Thus, a drill-down is annotated as the bridge from q_1 to q_2. 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):

$$\gamma^{year \to month}_{sum(cs\_quantity)}(\pi_{cs\_quantity}(\gamma^{date \to year}_{sum(cs\_quantity)}(\gamma^{store \to state}_{sum(cs\_quantity)}(\sigma_{cs\_product='1'}(raw\_data)))))$$

Finally, it is easy to see that query expressions q_1, q_2 and q_3 can be considered as constituting a session $s = \langle q'_1, q'_2, q'_3 \rangle$ over cube Sales*, and they can be modeled as unevaluated collections of fragments:*

$$q'_1 = \langle\langle allYears, state, allBranches, allStates, allLines\rangle, \{product = 1, TRUE_{YEAR}, \ldots, TRUE_{STATE}\}, \{cs\_quantity\}\rangle$$

$$q'_2 = \langle\langle month, state, allBranches, allStates, allLines\rangle, \{product = 1, TRUE_{YEAR}, \ldots, TRUE_{STATE}\}, \{cs\_quantity\}\rangle$$

$$q'_3 = \langle\langle month, state, allBranches, allStates, allLines\rangle, \{product = 1, TRUE_{YEAR}, \ldots, region =' SE'\}, \{cs\_quantity\}\rangle$$

4.5 Conclusion

This chapter introduced a technique for constructing a log as a set of sessions of OLAP queries, respecting the model presented in Chapter 3, from an ordered set of SQL queries. The techniques consists of normalizing multidimensional expressions, and checking for all pairs of successive such expressions, if a small number of OLAP operations can be used to transform the output of the former into the output of the latter. If the queries are not expressed in the multidimensional algebra, they are translated into it. A preliminary implementation is introduced in [106].

The next two chapters illustrate how such logs can be manipulated using a simple language inspired by the relational algebra.

Part III

Manipulating logs

Chapter 5

Languages for logs

This chapter proposes a manipulation language dedicated to query logs, stemming from the relational algebra. As this language is based on binary relations over sessions, such binary relation over sessions are first introduced in Section 5.1. Then, Section 5.2 introduces formally the language, and Section 5.3 illustrates its use with various examples.

This chapter relies on materials published in [12], [40] and [11]. The work presented in this chapter is being carried out in the scope of the doctorate thesis of Julien Aligon.

5.1 Binary relations over sessions

This section introduced various useful binary relations over sessions, used in the definition of the operators of the language for manipulated logs.

5.1.1 Relations over queries

We start with relations over queries, that will subsequently be used to define relations over sessions. In this chapter, we focus on specialization relations, i.e., partial (reflexive, antisymmetric, transitive) orders [76]. Similarities will be considered in the next chapter. A preference relation over queries is introduced in Chapter 8. In what follows, $q \succeq q'$ denotes that a query q is more general than a query q'.

Specialization relation over partially evaluated queries

We recall that partially evaluated queries are sets of references that can be expressed as cross product of sets of members (see Section 3.2.3). The specialization relation over sets of references follows the roll-up relation, in the sense that a query is more general than another if the former can be seen as a roll-up of the latter. It can be intuitively defined as follows: a query q is more general than another query q' if for all the references of q' there exists a reference in q

that generalizes it, and each reference of q generalizes at least one reference of q'.

Definition 5.1.1 *Let q and q' be two queries over the same schema, defined as two sets of references. q is more general than q', noted $q \succeq q'$, if $\forall r' \in q', \exists r \in q$ with $r \succeq r'$, and $\forall r \in q, \exists r' \in q'$ with $r \succeq r'$.*

It is easy to see that this relation is indeed a partial order. We note $q^\top$ the query $\{\langle m_1^{All}, \ldots, m_n^{All}\rangle\}$, where, for each hierarchy h_i, m_i^{All} is the coarsest member of the hierarchy. Finally, we note $Q^\top = \{q^\top\}$.

The two following operations are used to define a common ancestor of two queries q and q' w.r.t. the specialization relation $\succeq$.

Definition 5.1.2 (Union operator over partially evaluated queries) *Given two queries $q_1 = R_1^1 \times \ldots \times R_n^1$ and $q_2 = R_1^2 \times \ldots \times R_n^2$, $q_1 \cup q_2$ is the query $q = (R_1^1 \cup R_1^2) \times \ldots \times (R_n^1 \cup R_n^2)$.*

Definition 5.1.3 (lca operator for partially evaluated queries) *Let $q = R_1 \times \ldots \times R_n$ be a query. For a set of members M in dimension D, let $lca(M) = \{m \in \pi^*(D) | \forall m' \in M, (m \succeq m') \wedge \nexists m'' \in \pi^*(D), (m \succeq m'' \wedge m'' \succeq m')\}$. Then $lca(q) = lca(R_1) \times \ldots \times lca(R_n)$.*

A common ancestor to q and q' with respect to $\succeq$ is $lca(q, q') = lca(q \cup q')$.

Example 5.1.1 *Consider the following queries:*
$q_1 = \{LosAngeles\} \times \{AllRaces\} \times \{2001\} \times \{AllOccs\} \times \{Female\}$
$q_2 = \{California\} \times \{AllRaces\} \times \{2002\} \times \{AllOccs\} \times \{Female, Male\}$.
A common ancestor with respect to $\succeq$ is:
$q_3 = \{California\} \times \{AllRaces\} \times \{AllYears\} \times \{AllOccs\} \times \{AllSexes\}$.

Specialization relation over unevaluated queries

We recall that unevaluated queries are defined as a triple $\langle G, P, M\rangle$ (see Section 3.2.2). Given a schema $\langle L, H, Meas\rangle$, the group by sets, selection predicates and sets of measures of the queries that can be expressed over this schema, can all be respectively arranged into a lattice. The operations for manipulating unevaluated queries leverage this property.

Considering two group-by sets G and G', recall that we note $G \succeq_H G'$ if G is more general[1] then G' in the sense of the group-by lattice, whose top element $G^\top$ is the coarsest group by set. We define the *supp* and *inf* operations on group-by sets, to be, $supp(G, G') = min_{\succeq_H}\{G'' \in Dom(H) | G'' \succeq_H G \text{ and } G'' \succeq_H G'\}$ and $inf(G, G') = max_{\succeq_H}\{G'' \in Dom(H) | G \succeq_H G'' \text{ and } G' \succeq_H G''\}$.

For two sets of selection predicates P, P', we note $P \succeq_p P'$ if P is more selective than P', i.e., $val(P) \subseteq val(P')$, where $val(X)$ is the set of values used in predicate X. Note that, whatever $p_i \in P$, it is $val(p_i) \subseteq \{TRUE_i\}$, $val(p_i) \cap \{TRUE_i\} = val(p_i)$ and $val(p_i) \cup \{TRUE_i\} = \{TRUE_i\}$. We define the *supp*

[1] Note that G can also be interpreted as being more selective as G', in the sense that it asks for less values of the hierarchies to be retrieved.

and inf operations on sets of predicates, to be, $supp(P, P') = val(P) \cap val(P')$ and $inf(P, P') = val(P) \cup val(P')$. We recall that predicates in P (respectively P') are interpreted as a conjunction of Boolean predicates that indicate the values selected. Therefore, a query $\langle G, P, M\rangle$ where P features no selection predicate for a hierarchy h_i, is interpreted as a query selecting nothing, i.e., the unsatisfiable query. We thus define $P^\top$ as the set of sets of predicates where there exists one such hierarchy, i.e., $P^\top = \{P | P$ is a set of selection predicates for a schema $\langle L, H, M\rangle$ with n hierarchies, and $|P| < n\}$.

For two measure sets M, M', we note $M \succeq_m M'$ if $M \subseteq M'$. We define the $supp$ and inf operations on measure sets, to be, $supp(M, M') = M \cap M'$ and $inf(M, M') = M \cup M'$.

Given a schema $\langle L, H, Meas\rangle$, we consider the set of queries over this schema that are the less informative ones, that we note $Q^\top = \{\langle G, P, M\rangle | G = G^\top \vee P \in P^\top \vee M = \emptyset\}$. Among the queries in that set, a particular one is $q^\top = \langle G^\top, \emptyset, \emptyset\rangle$. Note that by less informative, we mean that its expression may not reflect a user's intention. For instance, a query $\langle G, P, \emptyset\rangle$ is interpreted as having no measure, thus nothing can be said about the user's intention in terms of analyzing facts.

We can now define the specialization relation over queries.

Definition 5.1.4 (Specialization relation over unevaluated queries) *Let $q = \langle G, P, M\rangle$ and $q' = \langle G', P', M'\rangle$ be two queries over the same schema. q is more general than q', noted $q \succeq q'$, if $G \succeq_H G'$ and $P \succeq_p P'$ and $M \succeq_m M'$.*

Given a schema $\langle L, H, Meas\rangle$, the set of queries over this schema together with the specialization relation $\succeq$ over queries form a lattice (being the product of three lattices). In particular, let $q = \langle G, P, M\rangle$ and $q' = \langle G', P', M'\rangle$ be two queries. Their most specific common ancestor is:

$$supp(q, q') = \langle supp_{\succeq_H}(G, G'), supp_{\succeq_p}(P, P'), supp_{\succeq_M}(M, M')\rangle$$

Example 5.1.2 *Consider the following queries over the* CENSUS *schema of Example 3.1.1:*

$$\begin{aligned}
q_1 =& \langle\langle City, AllRaces, Year, AllOccs, AllSexes\rangle,\\
& \{Region \in \{Pacific, Atlantic\}, Year = \text{"2001"}, TRUE_{RACE},\\
& TRUE_{OCCUPATION}, TRUE_{SEX}\},\\
& \{AvgIncome\}\rangle\\
q_2 =& \langle\langle Region, AllRaces, Year, AllOccs, Sex\rangle,\\
& \{Sex \in \{Male, Female\}, Year = \text{"2001"}, TRUE_{RACE},\\
& TRUE_{RESIDENCE}, TRUE_{OCCUPATION}\},\\
& \{AvgIncome\}\rangle
\end{aligned}$$

Their most specific common ancestor with respect to $\succeq$ is

$q_3 = \langle \langle Region, AllRaces, Year, AllOccs, AllSexes \rangle,$
$\{Year = \text{"2001"}, TRUE_{RACE}, Sex \in \{Male, Female\}$
$Region \in \{Pacific, Atlantic\}, TRUE_{OCCUPATION}\},$
$\{AvgIncome\}\rangle$

5.1.2 Relations over sessions

As for relations over queries, in this chapter, we focus on specialization relations over sessions. However, note that the relations over sessions are defined based on a relation $\succeq$ over queries whose semantics is not fixed. For instance, interpreting $\succeq$ as a preference relation over queries in what follows, allows to define preference relations over sessions.

In what follows, $s \succeq s'$ denotes that a session s is more general than a session s'.

Specialization relation over sessions

We recall that a session is a sequence of queries. In particular, we note $S^\top$ the set of sessions containing a query in $Q^\top$.

We introduce a specialization relation over sessions that is based on a specialization relation over queries.

Definition 5.1.5 (Specialization relation over sessions) *A session $s = \langle q_1, \ldots, q_{n_s} \rangle$ is more general than another session $s' = \langle q'_1, \ldots, q'_{n_{s'}} \rangle$, written $s \succeq s'$, if there exists a sequence of $n_{s'}$ integers $\{i_1, \ldots, i_{n_{s'}}\}$ with $i_1 = 1$, $i_{n_{s'}} = n_s$ and, for $k \in \{1, n_{s'} - 1\}$, $i_k \leq i_{k+1}$, such that, for all $j \in \{1, \ldots, n_{s'}\}$, it is $q_{i_j} \succeq q'_j$.*

The intuition of this specialization relation is given Figure 5.1. Note that if $s \succeq s'$ then the sequence of integers $\{i_1, \ldots, i_{n_{s'}}\}$ defines a partition $P = \{p_1, \ldots, p_{n_s}\}$ of $queries(s')$ where the p_i's can be ordered according to the queries of s, and the queries in each p_i, each less general than q_i, constitute a sub-session of s'. Given an integer n and a session s with $n \leq |s|$, we call a n-partition of s, a partition $P = \{p_1, \ldots, p_n\}$ of $queries(s)$ such that, for all $i \in [1, n-1]$, all queries of p_i precede the queries in p_{i+1} in s.

Example 5.1.3 *Consider queries q_1, q_2, q_3 of Example 5.1.2, q_3 being an ancestor of both q_1 and q_2, and sessions $s_1 = \langle q_1, q_2, q_1 \rangle$, $s_2 = \langle q_1, q_3 \rangle$ and $s_3 = \langle q_3 \rangle$. Then we have $s_3 \succeq s_2 \succeq s_1$.*

It can easily be seen that $\succeq$ is a specialization relation, being a partial order. However, note that with this specialization relation, sessions cannot be arranged into a lattice. In particular, there is more than one more specific common ancestor to a pair of sessions, as illustrated by the following example.

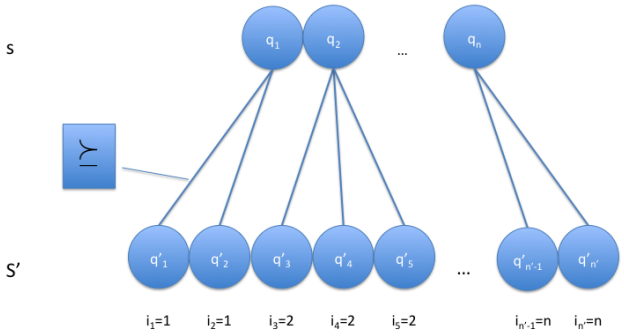

Figure 5.1: Specialization relation over sessions

Example 5.1.4 *For the sake of simplicity, we restrict queries to their measure sets. Consider sessions* $s = \langle\{a, c\}, \{a, b\}, \{b, d\}\rangle$, $s' = \langle\{a, c\}, \{b, d\}\rangle$. *These sessions have two more specific common ancestors:* $s'' = \langle\{a, c\}, \{b\}\rangle$ *and* $s''' = \langle\{a\}, \{b, d\}\rangle$.

Finding the most specific ancestors to a pair of sessions s, s' can be done as follows. First, note that the most specific ancestors of s, s', assuming s the shortest, have the length of s, noted $|s|$. The most specific ancestors can thus be obtained by computing all the $|s|$-partitions of s' and combining each such partition with s. More formally, let $s = \langle q_1, \ldots, q_n\rangle$ a session and s' another session such that $|s| \leq |s'|$. The most specific ancestors to s and s' are $anc(s, s') = \{\langle supp(q_1, p_1), \ldots, supp(q_n, p_n)\rangle | \{p_1, \ldots, p_n\}$ is a $|s|$-partition of $s'\}$. Note that, the cardinality of this set is at most $\binom{|s'|-1}{|s|-1}$, which is the number of $|s|$-partitions of s'. Note that, from this set, we may want to remove the sessions in $S^\top$.

Similarity relation over sessions

Suppose a function *dist* applied to pairs of sessions, giving the distance between two sessions. Similarity relations over sessions can be defined from such a function. For instance, $sim(s, s') \equiv dist(s, s') \leq \alpha$ for some real α indicates that two sessions s and s' are similar if their distance is below a threshold.

As another example, given a set S of sessions, and for some threshold α, $sim(s_0, s_n) \equiv \exists s_1, \ldots, s_{n-1} \in S, \forall i \in [0, n-1], dist(s_i, s_{i+1}) \leq \alpha$ indicates that two sessions s_0 and s_n are similar if there exists a chain of sessions with distance under α that can connect the two sessions. In that case, this relation is an equivalence relation over S.

Some distances between sessions are studied in Chapter 6.

5.2 A relational language for manipulating logs

5.2.1 Intuitions

The language introduced below enables to manipulate logs as defined in Chapter 3, i.e., sets of sessions. It is essentially an adaptation of the relational algebra with one main difference. The (extended) relational algebra enables the manipulation of relations, i.e., sets of tuples, that have a schema. The language introduced here manipulates logs, i.e., sets of sessions, that does not have a schema. The schema can therefore not be used in the expressions formed with this language. Instead, the operations of the language are parametrized by a binary relation over sessions θ, that may for instance be used to compare sessions. In particular, θ can be one of the specialization relations introduced above, or one of the similarity relations over sessions defined from distances that are presented in chapter 6.

5.2.2 Formal definitions

For the sake of simplicity, the names and symbols of the operations are the same as the ones used in the relational algebra. The language features 5 operations. Two of them are unary operations: selection σ and group by and aggregation π. Three of them are binary: join $\bowtie$, union $\cup$ and difference $\setminus$.

We now introduce the formal definitions, that we illustrate with simple examples. Advanced manipulations are given Section 5.3. In what follows, we consider a relation *sim* that relates two sessions if their similarity, measured with one of the similarity measures defined above exceeds a given threshold, or if they can be connected through a chain of sessions as defined in Section 5.1.2.

Selection

The selection operation enables to select from a log those sessions satisfying a given condition. This condition is given under the form of a relation to another session.

Definition 5.2.1 *Let L be a log, s be a session and θ be a binary relation over sessions.*

$$\sigma_{\theta,s}(L) = \{s' \in L | \theta(s, s')\}$$

Example 5.2.1 *The log can be searched for sessions similar to session s, using the sim relation, with expression: $\sigma_{sim,s}(L)$. This expression is called $findSimilar(s, sim, L)$ in what follows.*

The log can be searched for sessions containing a particular query q with the expression: $\sigma_{in,\langle q\rangle}(L)$ where $in(\langle q\rangle, s')$ is true if $q \in queries(s')$. In particular, this expression can be used to find sessions in $S^\top$ if $q = q^\top$. This expression is called $findS^\top(L)$ in what follows.

Set operations

As logs are defined as sets of sessions, the set union and set difference operations can be used to manipulate logs. The definitions are straightforward:

Definition 5.2.2 *Let L and L' be two logs.*

$$L \cup L' = \{s \in L \text{ or } s \in L'\} \text{ and } L \setminus L' = \{s \in L | s \notin L'\}.$$

Example 5.2.2 *Using the $\succeq$ specialization relation over sessions, a log L can be searched for all sessions that are more specific than another session s, with expression: $L \setminus \sigma_{\succeq,s}(L)$.*

Join

The join operation enables to combine the sessions in two logs, that are related through a relation θ.

Definition 5.2.3 *Let L and L' be logs, f be a binary functions outputting a session and θ be a binary relation over sessions.*

$$L \bowtie_{\theta,f} L' = \{f(s,s') | s \in L, s' \in L', \theta(s,s')\}.$$

Example 5.2.3 *Two logs can be compared using the join operation. For instance, given two logs L and L', the sessions that are similar among the logs can be found with the expression: $L \bowtie_{sim,first} L' \cup L' \bowtie_{sim,first} L$ where $first(s,s') = s$ for any pair of sessions s, s'.*

A log L can be searched for the best sessions it contains with respect to a partial order $\succeq$, i.e., $L \setminus (L \bowtie_{\succeq,second} L)$, with $second(s,s') = s'$. This expression is called $bestSessions(\succeq, L)$ in what follows.

Note that intersection can be simulated with the join operation. Indeed, $L \cap L' = L \bowtie_{same,first} L'$ with, for any two sessions s, s', $first(s,s') = s$ and $same(s,s') \equiv (s = s')$.

Note also that this join operation is not symmetric unless θ is symmetric and $f(s,s') = f(s',s)$ for all s, s'. Moreover, the f function is needed for closeness reason. In that sense, it is inspired by the join operation defined in [5] for joining cubes.

Grouping and aggregation

This operator allows to group sessions that are related to one another through relation θ, and to aggregate them using an aggregation function agg.

Definition 5.2.4 *Let L be a log, agg be a function aggregating a set of sessions into a session, and θ be a binary relation over sessions.*

$$\pi_{\theta,agg}(L) = \{agg(\{s' \in L | \theta(s,s')\}) | s \in L\}.$$

Example 5.2.4 *This operation can be used to transform sessions. For instance, extracting from a log L the last query of each session can be expressed by: $\pi_{same,last}(L)$ where, for any two s, s', it is $same(s,s') \equiv (s = s')$, and*

$last(\{\langle q_1, \ldots, q_n \rangle\}) = \langle q_n \rangle$. This expression is called extractLast(L) in what follows. If removeLast($\{\langle q_1, \ldots, q_n \rangle\}) = \langle q_1, \ldots, q_{n-1} \rangle$, is used instead of last, then the expression modifies each session of L by removing the last query. This expression will be called extractAllButLast(L) in what follows.

This operation can also be used for extracting the best queries from a log L in the sense of a partial order relation $\succeq$ over queries: $\pi_{better,getBest}(L)$ where, better(s, s') holds only if s contains at least one of the best queries of L[2] and s' does not, and getBest$(S) = \langle q_1, \ldots, q_n \rangle$ such that the q_i's are such best queries. This expression is called bestQueries$(\succeq, L)$ in what follows.

5.2.3 Properties

We briefly review the properties of the language in terms of closeness, completeness and minimality.

It can easily be seen that the language is closed under composition, the result of any operation being a set of sessions.

The language is complete. Indeed, for any pair of logs L and L', there is an expression that enables to transform L into L'. The transformation would proceed as follows:

a) pick a session s in L with σ,

b) transform it into a session s' of L' using π,

c) repeat the previous steps until all sessions of L' are formed,

d) union all transformed sessions to form L'.

The language is not minimal since σ can be simulated with $\bowtie$. Indeed, $\sigma_{\theta,s}(L) = L' \bowtie_{\theta,f} L$ where $L' = \{s\}$ and $f(s, s') = s'$. It can easily be seen that removing σ makes the language minimal.

5.3 Advanced manipulations

We close these chapter by illustrating the language with some expressions describing various user-centric tasks.

5.3.1 Summarizing and generalizing a log

The group by and aggregate operation can be used to summarize a log. For instance, given the equivalence relation *sim* defined in Section 5.1.2, sessions can be grouped together using *sim*, and then aggregated using the session in each group that minimizes the sum of the distances to other sessions of the group, with expression: $\pi_{sim,rep}(L)$, where $rep(S) = argmin_{1,s \in S, s' \in S}(\sum_S dist(s, s'))$, *dist* being one of the distances introduced above used in the definition of *sim*.

A log can be generalized by computing the most specific ancestors of the sessions it contains. The expression $E_1 = L \bowtie_{all,anc} L$, where $all(s, s')$ holds for

[2] i.e., $q \in L$ and $\nexists q' \in L$ with $q' \succeq q$

all $s, s' \in L$, gives the most specific ancestors of all pairs of sessions in L. Then $E_2 = E_1 \backslash findS^{\top}(E_1)$ removes the sessions in $S^{\top}$. Finally, $bestSessions(E_2, \succeq)$, gives the most general sessions of E_2. Calling this expression $generalize(L)$, more general logs can be expressed by: $generalize(\ldots(generalize(L)\ldots)$.

5.3.2 Personalization

Obviously, the language can be used to express dominance queries. For instance, assuming a preference relation $pref_s$ over sessions, extracting the preferred sessions from a log L is: $bestSessions(pref_s, L)$. Besides, assuming a preference relation $pref_q$ over queries, extracting the preferred queries from a log L is: $bestQueries(pref_q, L)$.

Obtaining a total order over queries from a total order over sessions can be expressed in the language. Assuming a total order $pref$ over sessions, the queries of a log can be ordered in a singleton where the order of queries in the session reflects the order over sessions. More precisely, this is achieved with $\pi_{all,agg}(L)$ with $all(s, s')$ holds for all $s, s' \in L$, and $agg(S) = \langle q_1, \ldots, q_n \rangle$ where for all $i, j \in \{1, n\}$, it is $i < j$ if either q_i and q_j belong to the same session s of S and $s^{-1}(q_i) < s^{-1}(q_j)$, or, they belong to different sessions s and s' respectively, and $pref(s, s')$ holds. This expression is called $order(agg, L)$.

Finally, note that a partial order over queries can be expressed for the queries of a log L as follows. First, separate all queries of the log by breaking all the sessions: $L' = extractLast(L) \cup extractLast(extractAllButLast(L)) \cup \ldots$ Then, use the join operation to form pairs of queries (i.e., sessions of length 2) to represent the order over the queries of L': $L' \bowtie_{\theta,f} L'$ with $\theta(\langle q \rangle, \langle q' \rangle)$ if a given relation between q and q' holds (like for instance $q \succeq q'$ or $q \subseteq q'$) and $f(\langle q \rangle, \langle q' \rangle) = \langle q, q' \rangle$.

5.3.3 Query recommendation

The language can be used to describe various query recommendation approaches. For instance, in the approach described in [37], the last queries of past sessions that are similar to a given session, are recommended. This approach can be expressed, for a log L, by: $extractLast(findSimilar(s, \theta, L))$ where θ is based on an extension of the edit distance to compare sessions (see next chapter). Moreover, these recommendations can be ordered using the expression $order(agg, L)$ defined above, where agg uses a total order on sessions derived from a total order on queries.

5.4 Conclusion

In this chapter, we introduced a log manipulation language inspired from the relational algebra, and relying on binary relations over queries and sessions. This language allows to manipulate logs in a flexible way, and can be used to search, filter, compare, combine, modify logs, and express advanced manipulations like summarization or query recommendation.

At the core of this language are binary relations over sessions. We have introduced at the beginning of this chapter some specialization relations over queries and sessions. The next chapter focuses on similarity measures over queries and sessions, for defining other relations to be used for log manipulation. More relations, especially preference relations over queries, will be considered in Part IV.

Chapter 6

Comparing sessions

This chapter introduces some similarities that will subsequently be used over the multidimensional objects to be found in query logs. Note that we do not address the comparison of multidimensional data since a comprehensive survey of similarities that can be used over OLAP data exists in the literature [15]. We focus on similarity measures for multidimensional queries and sessions. Section 6.1 first presents the approaches found in the literature for comparing queries and sequences. Then desirable properties of similarity measures for queries and sessions are listed in Section 6.2. Finally, Sections 6.3 and 6.4 introduce original similarity measures for comparing queries and sessions, respectively.

This chapter relies on [37] and on material developed in an ongoing work with Universita di Bologna, which is currently submitted [10].

6.1 Approaches for comparing queries and sessions

6.1.1 Comparing queries

We can distinguish two main motivations for comparing database queries. The first one is query optimization, where a query q to be evaluated is compared to another query q', with the goal of finding a better way of evaluating q. This motivation attracted a lot of attention, and covers classical problems like view usability [34, 48], query containment [2], plan selection [35], view selection [13, 42], data prefetching [93]. The second, more recent, motivation, is to suggest a query to the user without focusing on its evaluation. In this context, a query is compared to another one with the goal of helping the user exploring or analyzing a database. This includes query completion [109] and query recommendation [101, 24, 8, 37].

From a technical point of view, the approaches found in the literature can be classified according to (i) the *query model* they adopt, i.e., the structure used to represent queries; (ii) the *information source* from which the model of each query is derived; and (iii) the *function* used to compute similarity.

As seen in Chapter 3, query models range from uninterpreted text to the set of tuples being the query's answer.

As to the information source, it can be the query expression, e.g., the uninterpreted query text [110] or the list of query fragments (selection predicates, projection, etc.) [34, 109]. When fragments are used, only some of them may be taken into account; for instance, only the selection attributes are used in [6, 109] whereas all fragments are used in [34, 48]. The information source can also be related to the database queried; more precisely, it can be:

- the database instance, e.g., the query result or the active domain of the database attributes [6, 24, 37, 101]; In the former case, the query can be either fully [101] or partially [37] evaluated;
- the statistics used by the query optimizer, like table sizes and attribute cardinalities [35];
- the database schema, e.g., the keys defined or the index used to process a selection [35, 42];
- the query log, if the query model relies on other queries that have previously been launched on the same database. For instance, in [24, 8, 13, 101] a query is modeled in terms of its links with other queries or how many times it appears in the log.

Finally, the result of query comparison can be a Boolean or a score, usually normalized in the [0..1] interval. The first case applies when queries are tested for equivalence [2] or view adaptation [48], or when the goal is to group queries based on some criteria [93, 109]. In this case, the comparison can be a simple equality test of the query schemata [93, 109] or it can be based on separate tests of query fragments [48]. In the second case, the comparison is normally based on classical functions applied to the query schemata. For instance, if the query is modeled as a vector, cosine [6, 8, 24], inner product [101], or Hamming distance [13] can be used; if the query is modeled as a set, the Jaccard index [101] or the Hausdorff distance [37] can be used. Sometimes, more sophisticated similarity functions are used. For instance, [110] uses a measure based on entropy to cluster queries modelled as strings. In [42], similarity between OLAP queries is computed based on the relative position of the query group-by sets within the group-by lattice.

Table 6.1 summarizes the approaches reviewed in this section. Note that [101] proposes two ways of comparing queries: (1) based on the frequency of the query in the log, and (2) based on the query result. Letters S, P, and C indicate the fragments used by the approach (S for selection, P for generalized projection —including the group-by set and the aggregation operator—, and C for cross-product).

6.1.2 Comparing sequences

Comparing sequences has attracted a lot of attention especially in the context of string processing, with applications like information retrieval, spell-checkers,

Ref.	*Motivation*	*Model*	*Source*	*Similarity Function*
[48]	optimization	sets	S, P, C	fragment tests
[24]	recommendation	vector	db instance, log	cosine
[8]	recommendation	vector	S, P, log	cosine
[6]	optimization	vector	S, db instance	cosine
[13]	optimization	vector	S, P, log	Hamming distance
[35]	optimization	vector	S, C, db statistics	Hamming distance
[101] (1)	recommendation	vector	log	inner product
[101] (2)	recommendation	set	db instance	Jaccard index
[37]	recommendation	set	db instance	Hausdorff distance
[93]	optimization	sets	S, P	query schema equality
[42]	optimization	set	P, db schema, db statistics	group-by lattice
[110]	recommendation	string	SQL sentence	entropy
[109]	recommendation	graph	S, P, C	query schema equality

Table 6.1: Query comparison approaches at a glance

bioinformatics, and record linkage [29, 78]. The existing approaches are inspired by different principles.

In *token-based approaches* sequences are treated as bags of elements, and classical set similarity measures like Jaccard and Hausdorff, and all their variants, can be used or adapted. Of course, these approaches are not sensible to the order of sequence elements. When the sequences to be compared are taken from a *corpus*, the popular *term frequency-inverse document frequency* (Tf-Idf) weight can be adopted, which weighs each element of a sequence using (positively) their frequency in the sequence and (negatively) their frequency in the corpus. A cosine is then used to measure the similarity between two vectors of weights.

Other approaches compare two sequences by comparing their subsequences. A basic approach here is to use the size of the longest common subsequence (LCS).[1] An approach often used in statistical natural language processing relies on n-grams, i.e., substrings of size n of a given sequence. A popular similarity measure using n-grams is the *Dice coefficient*, an extension of the Jaccard index defined as twice the number of shared n-grams over the total number of n-grams:

$$Sim_{Dice}(s, s') = \frac{2|ngrams(s) \cap ngrams(s')|}{|ngrams(s)| + |ngrams(s')|}$$

Other approaches compare sequences based on their *edit distance*, i.e., in terms of the cost of the atomic operations necessary to transform one sequence into another. Many edit distances have been proposed that differ on the number, type, and cost of the edit operations. The most popular are the *Levenshtein distance*, that allows insert, delete, and substitute, and the *sequence alignment distance*, that allows match, replace, delete, and insert [29, 82].

Finally, in *two-level* approaches sequences are compared based on the similarity between their elements. A simple example is the Hausdorff distance between sets, that relies on the distance between elements of the set. In [77] the similarity between sequences s and s' is the average of the highest similarities

[1] Note that, while substrings are consecutive parts of a string, subsequences need not be.

between pairs of elements of s and s':

$$Sim_{M\&E}(s, s') = \frac{1}{|s|} \sum_{s_i \in s} max_{s'_j \in s'} \{Sim_{elem}(s_i, s'_j)\}$$

where Sim_{elem} measures the similarity between single elements[2]. In *soft Tf-Idf* [29, 78], the Tf-Idf weight is extended using the similarity of sequence elements; more precisely,

$$Sim_{soft}(s, s') = \sum_{s_i \in Close_\theta(s,s')} T(s_i, s) \cdot T(s_i, s') \cdot max_{s'_j \in s'} \{Sim_{elem}(s_i, s'_j)\}$$

where $T(s_i, s)$ is a normalized form of the Tf-Idf of element s_i within sequence s, θ is a threshold, and $Close_\theta(s, s')$ is the set of elements $s_i \in s$ such that there is at least an element $s'_j \in s'$ with $Sim_{elem}(s_i, s'_j) > \theta$. While the two previous two-level approaches do not consider the ordering of elements within sequences, the *Smith-Waterman algorithm* relies on element ordering; it can be used to efficiently find the best alignment between subsequences of two given sequences by ignoring the non-matching parts of the sequences [100]. It is a dynamic programming algorithm based on a matrix H whose value in position (i, j) expresses the score for aligning subsequences of s and s' that end in elements s_i and s'_j, respectively. The similarity between two sequences is then measured as the highest alignment score in the matrix. This matrix is recursively defined based on the following formula:

$$H(i, j) = max \left\{ \begin{array}{c} 0; \\ H(i-1, j-1) + Sim_{elem}(s_i, s'_j); \\ max_{k \geq 1}\{H(i-k, j) - c_k\}; \\ max_{k \geq 1}\{H(i, j-k) - c_k\} \end{array} \right\}$$

where c_k is the cost of introducing a gap of length k in the matching between s and s'. Note that, here, the similarity between two elements can be negative, to express that there is a mismatch between them; intuitively, the algorithm seeks an optimal trade-off between the cost for introducing a gap in the matching subsequences and the cost for including a poorly matching pair of elements.

It should be noted that these approaches measure the sequence similarity (or distance). Tf-idf and Dice coefficient are normalized in [0,1] while Edit distance and sequence alignment are usually not (but can easily be) normalized.

6.2 Requirements for similarity measures for OLAP sessions

The goal of this section is to list a number of requirements to be used for (i) understanding which approaches, among all those proposed in the literature for query and sequence comparison, are eligible for the OLAP context; and

[2] M & E refer to the authors name.

(ii) driving the adaptation and extension of the eligible approaches towards the development of an original approach to OLAP session comparison.

We start by proposing a first set of requirements, suggested by the specific features of the OLAP context and by our experience in the field:

♯1 Multidimensional databases store huge amounts of data, and OLAP queries may easily return large volumes of results. To adequately address the efficiency issue, we compute similarity at the intensional level, i.e., considering only query expressions and neglecting all the extensional aspects (such as the actual query results).

♯2 It is unlikely that two OLAP sessions share identical queries; this feature is better managed by having comparisons of single queries result in a score rather than in a Boolean.

♯3 A typical OLAP query is defined by the fact to be analyzed, one or more measures to be computed, a set of hierarchy levels for aggregating measure values, a predicate for filtering a subset of events, and a presentation. Though the presentation chosen for displaying the results of an OLAP query (e.g., a cross-tab or a pie-chart) certainly has an influence on how easily users can interpret these results, it does not affect the actual informative content, so it should not be considered when comparing queries.

To discover additional requirements for OLAP sessions similarity, we prepared a questionnaire asking to give a qualitative evaluation of the similarity between couples of OLAP queries and couples of OLAP sessions over a simple multidimensional schema. The questionnaire was submitted to all the teachers and PhD students of the First European Business Intelligence Summer School (eBISS 2011)[3], as well as to the master students of two specialistic courses on data warehouse design at the Universities of Bologna (Italy) and Tours (France). Overall, 41 answers were collected. The additional requirements emerging from an analysis of the questionnaire results can be summarized as follows:

♯4 The selection predicate is the most relevant component in determining the similarity between two OLAP queries, followed by the group-by set. The less important component is the set of measures to be returned.

♯5 The order of queries is relevant in determining the similarity between two sessions, i.e., two sessions sharing the same queries but in different orders have low similarity.

♯6 Recent queries are more relevant than old queries in determining the similarity between two OLAP sessions.

♯7 The longest the matching section of two sessions, the highest their similarity.

♯8 Two sessions that match with one or more gaps (one or more non-matching queries are present) are similar, but their similarity is lower than the one of two sessions that match with no gaps.

[3] http://cs.ulb.ac.be/conferences/ebiss2011/

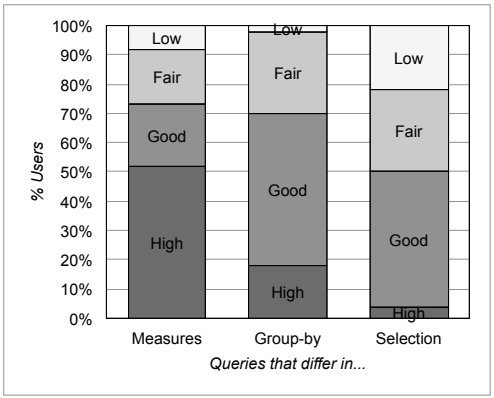

Figure 6.1: Perceived similarities for OLAP queries only differing in one of their three main components

In particular, as to point ♯4, in Figure 6.1 we show the percentages of users that perceive a given level of similarity for couples of queries that only differ in either their measure sets, or their selection predicates, or their group-bys. Apparently, measures are the less important component in determining similarity since most users perceive as highly similar two queries that only differ in their measures. The opposite holds for the selection predicate component.

6.3 Similarity measures for OLAP queries

This section introduce some similarity measures for OLAP queries.

6.3.1 Similarities for evaluated or partially evaluated queries

When queries expressions are evaluated or partially evaluated, similarities for multidimensional data, as reviewed in [15], can be used. We illustrate this with a distance between sets of references that we proposed in [37], based on the Hausdorff distance.

Queries being modeled as sets of references, comparing two queries boils down to comparing two sets of references which can indeed be done with the classical Hausdorff distance for comparing two sets, based on a distance between the elements of the sets. We first introduce the distance between references. This distance is based on the shortest path between members in a hierarchy, defined as follows: Given a cube C and one of its hierarchies H, the distance $\sigma_{members}$ between two members m_1, m_2 in this hierarchy is the length of the shortest path from m_1 to m_2 in H.

Definition 6.3.1 (Distance between references) *Given an n-dimensional cube C and two references $r_1 = \langle r_1^1, \ldots, r_1^n \rangle$ and $r_2 = \langle r_2^1, \ldots, r_2^n \rangle$, the distance between r_1 and r_2 is $\sigma_{sp}(r_1, r_2) = \sum_{i=1}^{n} \sigma_{members}(r_1^i, r_2^i)$.*

Definition 6.3.2 (Distance between sets of references) *Given two queries q_1 and q_2, the Hausdorff distance between q_1 and q_2 is:*

$$\sigma_H(q_1, q_2) = \max\{\, \max_{r_1 \in q_1} \min_{r_2 \in q_2} \sigma_{sp}(r_1, r_2), \\ \max_{r_2 \in q_2} \min_{r_1 \in q_1} \sigma_{sp}(r_1, r_2) \,\}$$

Example 6.3.1 *Consider the following queries:*

$q_1 = \{Pacific, Atlantic\} \times \{AllRaces\} \times \{2001\} \times \{AllOccs\} \times \{AllSexes\}$

$q_2 = \{AllCities\} \times \{AllRaces\} \times \{2002\} \times \{AllOccs\} \times \{Male, Female\}$

Their distance is $\sigma_H(q_1, q_2) = 1 + 0 + 2 + 0 + 1 = 4$.

This distance can be normalized in [0,1] by dividing by the maximum distance σ_H^{max}, which is, for a schema $\mathcal{M} = \langle A, H, M \rangle$ with $H = \{h_1, \ldots, h_n\}$, $\sigma_H^{max} = 2 \times \sum_{i=1}^{n} |lev(h_i)|$. Interestingly it is shown in [83] that this distance is a metric in the sense that it satisfies non-negativity, symmetry and triangle inequality.

6.3.2 Similarities for unevaluated queries

When queries are not evaluated, they can be modeled as set of fragments. In this context, we have proposed a measure tailored for OLAP queries, under the form of a function which is a combination of three components: one related to group-by sets, one to selection predicates, and one to measure sets.

To define group-by set similarity, we first introduce the notion of distance between levels in a hierarchy.

Definition 6.3.3 (Distance between hierarchy levels) *Let $\mathcal{M} = \langle A, H, M \rangle$ be a schema, $h_i \in H$ be a hierarchy, with $Lev(h_i) = \{l_0, \ldots, l_d\}$ and $l_0 \succeq_{h_i} l_1 \ldots \succeq_{h_i} l_d$, and $l_x, l_y \in Lev(h_i)$ be two levels. The* distance *between l_x and l_y, $Dist_{lev}(l_x, l_y)$, is the difference between the positions of l_x and l_y within the roll-up order $\succeq_{h_i}$, i.e., $|x - y|$.*

Definition 6.3.4 (Group-by set similarity) *Let q and q' be two queries, both over schema $\mathcal{M}$, with group-by sets g and g', respectively, and let $g.h_i$ ($g'.h_i$) denote the level of h_i included in g (g'). The* group-by set similarity *between q and q' is*

$$\sigma_{gbs}(q, q') = 1 - \frac{\sum_{i=1}^{n} \frac{Dist_{lev}(g.h_i, g'.h_i)}{|Lev(h_i)|-1}}{n}$$

where n is the number of hierarchies in $\mathcal{M}$.

Our definition of selection similarity takes into account both the attributes and the constants that form the selection predicate. In particular, for each hierarchy, two identical predicates are given maximum similarity, and non-identical predicates are given decreasing similarities according to the distance between the hierarchy levels they are expressed on.

Definition 6.3.5 (Distance between selection predicates) *Let $\mathcal{M} = \langle L, H, M \rangle$ be a schema, and p_i and p'_i be two selection predicates over hierarchy $h_i \in H$. Let $p_i.h_i \in Lev(h_i)$ denote the level of h_i involved in p_i (conventionally, $TRUE_i.h_i = ALL_i$). The* distance *between p_i and p'_i is*

$$Dist_{pred}(p_i, p'_i) = \begin{cases} 0, \text{ if } p_i = p'_i; \\ Dist_{lev}(p_i.h_i, p'_i.h_i) + 1, \text{ otherwise} \end{cases}$$

According to this definition, the distance between two selection predicates on h_i is 0 if they are expressed on the same level and the same value, 1 if they are defined on the same level but not on the same value. Note that, for the sake of simplicity, we consider here only predicates of the form $l = v$ (with l a level and v a value). Should predicates of the form $l \in V$ be also considered, a Jaccard index on sets of selected values could be used, in such a way that the distance of two predicates on the same level is a real in $[0, 1]$.

Definition 6.3.6 (Selection similarity) *Let q and q' be two queries, both over schema $\mathcal{M}$, with selection predicates $Pred$ and $Pred'$, respectively, with $Pred = \{p_1, \ldots, p_n\}$ and $Pred' = \{p'_1, \ldots, p'_n\}$. The* selection similarity *between q and q' is*

$$\sigma_{sel}(q, q') = 1 - \frac{\sum_{i=1}^{n} \frac{Dist_{pred}(p_i, p'_i)}{|Lev(h_i)|}}{n}$$

Finally, to define the measure similarity, we use the Jaccard index.

Definition 6.3.7 (Measure similarity) *Let q and q' be two queries, both over schema $\mathcal{M}$, with measure sets $Meas$ and $Meas'$, respectively. The* measure similarity *between q and q' is*

$$\sigma_{meas}(q, q') = \frac{|Meas \cap Meas'|}{|Meas \cup Meas'|}$$

We can now define the similarity between two OLAP queries as the weighted average of the three similarity components defined above.

Definition 6.3.8 (Similarity of unevaluated queries) *Let q and q' be two queries, both over schema $\mathcal{M}$. The* similarity *between q and q' is*

$$\sigma_{que}(q, q') = \alpha \cdot \sigma_{gbs}(q, q') + \beta \cdot \sigma_{sel}(q, q') + \gamma \cdot \sigma_{meas}(q, q')$$

where α, β, and γ are normalized to 1.

Example 6.3.2 *Let q_1 and q_3 be the queries of Example 3.2.4, i.e.,*

$$q_1 = \langle G^{\perp}, TRUE, \{\mathsf{AvgIncome}\} \rangle$$
$$q_3 = \langle G_3, \{\mathsf{Year} = 2011, TRUE_{\mathsf{RESIDENCE}}, \ldots, TRUE_{\mathsf{SEX}}\}, \{\mathsf{AvgIncome}\} \rangle$$

where

$$G^{\perp} = \langle \mathsf{City}, \mathsf{Race}, \mathsf{Year}, \mathsf{Occ}, \mathsf{Sex} \rangle$$
$$G_3 = \langle \mathsf{Region}, \mathsf{Race}, \mathsf{Year}, \mathsf{Occ}, \mathsf{Sex} \rangle$$

and $TRUE = \{TRUE_{\mathsf{RESIDENCE}}, TRUE_{\mathsf{RACE}}, TRUE_{\mathsf{TIME}}, TRUE_{\mathsf{OCCUPATION}}, TRUE_{\mathsf{SEX}}\}$.

The similarity between q_1 *and* q_3 *is computed as follows:*

$$\sigma_{gbs}(q_1, q_3) = 1 - \frac{(2/3 + 0/3 + 0/1 + 0/1 + 0/1)}{5} = 0.87$$
$$\sigma_{sel}(q_1, q_3) = 1 - \frac{(0/4 + 0/4 + 2/2 + 0/2 + 0/2)}{5} = 0.80$$
$$\sigma_{meas}(q_1, q_3) = 1$$

If priority is given to selections over group-by sets and measures, with $\alpha = 0.25$, $\beta = 0.5$, *and* $\gamma = 0.25$, *then* $\sigma_{que}(q_1, q_3) = 0.25 \times 0.87 + 0.5 \times 0.80 + 0.25 \times 1 = 0.87$.

6.4 Similarity measures for OLAP sessions

In this section, we introduce various two level techniques for comparing OLAP sessions by extending the classical sequence similarities presented Section 6.1.2. We will use the following toy log for illustrating them. For the sake of readability, the CENSUS schema is reduced to three hierarchies: RESIDENCE, RACE and TIME.

Example 6.4.1 *An example of OLAP session of length 3 on the* CENSUS *schema is* $s = \langle q_1, q_2, q_3 \rangle$, *where:*

$$q_1 = \langle G_1, \{TRUE_{\mathsf{RESIDENCE}}, TRUE_{\mathsf{RACE}}, TRUE_{\mathsf{TIME}}\}, \{\mathsf{AvgIncome}\} \rangle$$
$$q_2 = \langle G_2, \{TRUE_{\mathsf{RESIDENCE}}, TRUE_{\mathsf{RACE}}, TRUE_{\mathsf{TIME}}\}, \{\mathsf{AvgIncome}\} \rangle$$
$$q_3 = \langle G_2, \{TRUE_{\mathsf{RESIDENCE}}, TRUE_{\mathsf{RACE}}, (\mathsf{Year} = 2011)\}, \{\mathsf{AvgIncome}\} \rangle$$

where the group-by sets are:

$$G_1 = \langle \mathsf{City}, \mathsf{Race}, \mathsf{Year} \rangle$$
$$G_2 = \langle \mathsf{Region}, \mathsf{Race}, \mathsf{Year} \rangle$$
$$G_3 = \langle \mathsf{Region}, \mathsf{RaceGroup}, \mathsf{Year} \rangle$$

Note that the user applied a roll-up operator to move from q_1 *to* q_2, *and a slice operator to move from* q_2 *to* q_3.

6.4.1 An Extension of the Levenshtein Distance

The Levenshtein distance compares two strings in terms of the cost of the atomic operations (typically insertion, deletion, and substitution of a character) necessary to transform one string into another. Given two strings s and s' of v and v' characters, respectively, a $(v+1) \times (v'+1)$ distance matrix D of reals is recursively defined in terms of the deletion, insertion, and substitution costs; the Levenshtein distance between s and s' is found in the bottom-right cell of D, that represents the minimum sum of the operation costs to transform s in s'.

In the traditional formulation, an operation is applied in absence of a perfect match between the compared characters. In our case this is too restrictive, because OLAP queries cannot be treated as atomic terms. So we consider two queries as matching when their similarity is above a given threshold θ, and we apply a transformation operation when the similarity is under θ. Besides, we normalize distances using the length of the longest of the two sessions involved, so that the cost of a single mismatch is lower for longer sessions.

Definition 6.4.1 (Levenshtein Similarity of OLAP Sessions) *Let s and s' be two OLAP sessions on schema $\mathcal{M}$, of lengths v and v' respectively. Given a matching threshold θ, the distance matrix for s and s' is a $(v+1) \times (v'+1)$ matrix D of reals recursively defined as follows:*

$$D_\theta(i,j) = \begin{cases} 0, \text{ when } i = 0 \text{ or } j = 0 \\ D_\theta(i-1, j-1), \text{ when } i,j > 0 \text{ and } \sigma_{que}(s_i, s'_j) \geq \theta \\ min \left\{ \begin{array}{c} D_\theta(i-1,j)+1; \\ D_\theta(i,j-1)+1; \\ D_\theta(i-1,j-1)+1 \end{array} \right\}, \text{ when } i,j > 0 \text{ and } \sigma_{que}(s_i, s'_j) < \theta \end{cases}$$

where s_i is the i-th *query of session s. The* similarity *between s and s' is:*

$$\sigma_{Levenshtein}(s, s') = 1 - \frac{D_\theta(v, v')}{max\{v, v'\}}$$

Note that, like in most applications of the Levenshtein distance, all transformation costs are set to 1.[4] Note also that in [37], another approach is investigated, for partially evaluated queries, where, in the case of a mismatch, the cost of a substitution operation corresponds to the distance between queries.

Example 6.4.2 *Let a log consist of three sessions, $s = \langle q_1, q_2, q_3 \rangle$, $s' = \langle q_4, q_5, q_6, q_7 \rangle$, and $s'' = \langle q_8, q_9 \rangle$. Table 6.2 represents each single query in terms of our query model; the involved group-by sets are G_1, G_2, and G_3 of Example 6.4.1, while the selection predicates are:*

$$c_1 = \{TRUE_{\mathsf{RESIDENCE}}, TRUE_{\mathsf{RACE}}, TRUE_{\mathsf{TIME}}\}$$
$$c_2 = \{TRUE_{\mathsf{RESIDENCE}}, TRUE_{\mathsf{RACE}}, (\mathsf{Year} = 2011)\}$$
$$c_3 = \{TRUE_{\mathsf{RESIDENCE}}, (\mathsf{RaceGroup} = Chinese), TRUE_{\mathsf{TIME}}\}$$

[4] In the formula, the three rows of the *min* argument deal with deletions, insertions, and substitutions, respectively.

		Queries								
		q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9
Group-by set		G_1	G_2	G_2	G_1	G_2	G_3	G_3	G_1	G_1
Measures	AvgCostElect				✓	✓			✓	✓
	AvgIncome	✓	✓	✓	✓	✓	✓	✓	✓	
Selection predicates		c_1	c_1	c_2	c_3	c_3	c_1	c_2	c_1	c_1

Table 6.2: Queries for Example 6.4.2

	q_4	q_5	q_6	q_7
q_1	0.750	0.676	0.889	0.778
q_2	0.676	0.750	0.963	0.852
q_3	0.565	0.639	0.852	0.963

Table 6.3: Query similarities for Example 6.4.2

Table 6.3 shows the query similarities for sessions s and s' using $\theta = 0.7$. The minimum cost to transform s' to s is obtained by matching queries as follows: $\langle q_1, q_4\rangle, \langle q_2, q_6\rangle, \langle q_3, q_7\rangle$ and applying a deletion operation to q_5 since $\sigma_{que}(q_1, q_5) < \theta$. Thus, it is $\sigma_{Levenshtein}(s, s') = 1 - \frac{1}{4} = 0.75$.

6.4.2 An Extension of the Dice Coefficient

In the OLAP context, the concept of "shared" n-grams becomes that of "similar" n-grams. Two n-grams r and r' are similar if their queries are pairwise similar, i.e., if their similarity is above threshold θ. To ensure symmetry while being consistent with the original definition, in our two-level extension similarity is defined as follows.

Definition 6.4.2 (Dice Similarity of OLAP Sessions) *Let s and s' be two OLAP sessions on schema $\mathcal{M}$, and $n \geq 1$. The* similarity *between s and s' is*

$$\sigma_{Dice}(s, s') = \frac{2 \times min\{|SNgram_\theta(s, s')|, |SNgram_\theta(s', s)|\}}{|Ngram(s)| + |Ngram(s')|}$$

where $Ngram(s)$ is the set of n-grams of s and $SNgram_\theta(s, s') \subseteq Ngram(s)$ is the set of n-grams of s that have a similar n-gram in s', i.e., $SNgram_\theta(s, s') =$

$$\{r \in Ngram(s) | \exists r' \in Ngram(s'), \sigma_{que}(r_i, r'_i) \geq \theta \; \forall i = 1, \ldots, n\}$$

Example 6.4.3 *Applying the above definition to Example 6.4.2, with n=1, we obtain $\sigma_{Dice}(s, s') = \frac{2 \times min\{1,2\}}{1+2} = 0.67$.*

6.4.3 An Extension of Tf-Idf

In the Tf-Idf approach, the similarity between two sets of tokens (in information retrieval applications, tokens are lemmas and sets of tokens are documents) depends on both the frequency of each token in the sets and its frequency in a corpus. In our context, this approach can be adopted if the OLAP sessions to be compared are taken from a log, to penalize the queries that are more frequent in the log when assessing similarity.

To propose an extension we start with the definition of *soft Tf-Idf* [8], that introduces a concept of similarity between tokens:

$$sim_{soft}(s, s') = \sum_{s_i \in Close_\theta(s,s')} T(s_i, s) \cdot T(s'_{j_i}, s') \cdot sim_{token}(s_i, s'_{j_i})$$

where $T(s_i, s) = \frac{tfidf(s_i,s)}{\sqrt{\sum_{s_k} tfidf(s_k,s)^2}}$, sim_{token} measures the similarity between two tokens, $s'_{j_i} = argmax_{s'_j \in s'}\{sim_{token}(s_i, s'_j)\}$, θ is a threshold, and $Close_\theta(s, s')$ is the set of token $s_i \in s$ such that there is at least a token $s'_j \in s'$ with $sim_{token}(s_i, s'_j) > \theta$. This definition suffers from some drawbacks: (1) It uses the "crisp" definition of Tf-Idf in T, whereas in our context a "soft" version (i.e., one based on query similarity) should be used instead; (2) The soft Tf-Idf is not symmetric, which is not desirable for a similarity measure; (3) There may be more than one token s'_{j_i} in s' that maximizes sim_{token} with s_i; (4) There is a problem with counting that makes the similarity not normalized [78]. To cope with the first issue, we inject the similarity σ_{que} in the definition of Tf-Idf:

$$tfidf(s_i, s) = \frac{|Close_\theta(s_i, s)|}{\sum_{s_k \in Q} |Close_\theta(s_k, s)|} \cdot log \frac{|L|}{|\{s \in L | Close_\theta(s_i, s) \neq \emptyset\}|}$$

where L is the log, Q is the set of all queries in L, and $Close_\theta(s_i, s)$ is the set of queries of s that are similar to s_i. Symmetry can be achieved by modifying the definition of similarity to work on pairs of queries, each relating a query in one session with one of its closest queries in the other session; this set of pairs is defined by:

$$R_\theta(s, s') = \{\langle s_i, s'_k \rangle | s_i \in s, s'_k \in Closest_\theta(s_i, s'))\} \cup \\ \{\langle s_l, s'_j \rangle | s'_j \in s', s_l \in Closest_\theta(s'_j, s)\}$$

where $Closest_\theta(s_i, s)$ is the set of queries of s that, among those whose similarity with s_i is above θ, have maximum similarity. Note that some query in a session appears more than once in $R_\theta(s, s')$ if there is more than one query in the other session with maximum similarity. This solves the third issue. Finally, to cope with the fourth issue, the similarity is computed as the cosine of the two vectors obtained by taking the Tf-Idf of all the first (respectively, second) queries of the pairs.

Definition 6.4.3 (Tf-Idf Similarity of OLAP Sessions) *Let s and s' be two OLAP sessions on schema $\mathcal{M}$. The* similarity *between s and s' is*

$$\sigma_{log}(s, s') = \sum_{\langle s_i, s'_j \rangle \in R_\theta(s,s')} T(s_i, s, s') \times T(s'_j, s', s) \times \sigma_{que}(s_i, s'_j)$$

where

$$T(s_i, s, s') = \frac{tfidf(s_i, s)}{\sqrt{\sum_{\langle s_i, s'_j \rangle \in R_\theta(s,s')} tfidf(s_i, s)^2 + \sum_{Closest_\theta(s_i,s')=\emptyset} tfidf(s_i, s)^2}}$$

$$T(s'_j, s', s) = \frac{tfidf(s'_j, s')}{\sqrt{\sum_{\langle s_i, s'_j \rangle \in R_\theta(s,s')} tfidf(s'_j, s')^2 + \sum_{Closest_\theta(s'_j,s)=\emptyset} tfidf(s'_j, s')^2}}$$

Example 6.4.4 *The Tf-Idf similarity for sessions s and s' of Example 6.4.2 is 0.96.*

As any cosine similarity, σ_{log} can be turned into the angle distance $arcos(\sigma_{log})$, which is a metric [22].

6.4.4 An extension of sequence alignment

As emerged in Section 6.2, a comparison of OLAP sessions should support subsequence alignment, keep query ordering into account, and allow gaps in the matching subsequences. The Smith-Waterman algorithm mentioned in Section 6.1 has all these features. It relies on a distinction between *matching* elements (whose similarity is positive) and *mismatching* elements (whose similarity is negative), and is based on a matrix whose cells show the score for aligning two sequences starting from a specific couple of elements. Each score is the result of a trade-off between the cost for introducing a gap in the matching subsequences and the cost for including a mismatching pair of elements.

Unfortunately, none of the implementations available in the literature can be directly applied here for different reasons:

- The algorithm was originally aimed at molecular comparison, so sequence elements were taken from a set that is known a priori (the set of all amino acids). This allows matching and mismatching pairs to be enumerated and a similarity score to be assigned in advance to each possible couple of elements. In the OLAP context matching elements are queries, and the domain of the possible OLAP queries is huge (requirement ♯2); besides, the similarity between two queries is always positive, so separating matching and mismatching queries requires the adoption of a threshold.

- For the same reason mentioned above, in all previous implementations the cost for introducing a gap could be assigned in advance to each possible couple of elements. Conversely, in our case it must be determined at run-time based on the two specific sessions being compared (requirement ♯8).

- In all previous implementations all matchings were considered to be equally important, while in OLAP sessions a matching between recent queries should be given more relevance (requirement ♯6).

To address all these issues, we propose an extension of the Smith-Waterman algorithm that relies on the matrix defined below. The value in position (i, j)

of this matrix is a score that expresses how "well" two sessions s and s' match when they are aligned ending in queries s_i and s'_j. Intuitively, each score is recursively calculated by progressively adding the similarities between all pairs of matching queries in the two sessions. A *match threshold* θ is used to distinguish matches from mismatches; a *time-discounting* function $\rho(i, j)$ is used to promote alignments based on recent queries; finally, a *gap penalty* δ is used to discourage discontinuous alignments.

Definition 6.4.4 (OLAP Session Alignment Matrix) *Let s and s' be two OLAP sessions on schema $\mathcal{M}$, of lengths v and v' respectively. Given a match threshold θ, the* (OLAP session) alignment matrix *for s and s' is a $(v+1)\times(v'+1)$ matrix A of reals recursively defined as follows:*

$$A(i,j) = \begin{cases} 0, \text{ when } i = 0 \text{ or } j = 0 \\ max \left\{ \begin{array}{c} 0; \\ A(i-1, j-1) + (\sigma_{que}(s_i, s'_j) - \theta) \cdot \rho(v-i, v'-j); \\ max_{1 \le k < i}\{A(k,j) - \delta \cdot (i-k)\}; \\ max_{1 \le k < j}\{A(i,k) - \delta \cdot (j-k)\} \end{array} \right\}, \text{ else} \end{cases}$$

where δ is the average similarity between all couples of queries in s and s' whose similarity is above θ:

$$\delta = avg_{(i,j):\sigma_{que}(s_i,s'_j) \ge \theta}\{\sigma_{que}(s_i, s'_j)\} \ ,$$

ρ is a two-dimensional logistic sigmoid function:

$$\rho(i,j) = 1 - \frac{1 - \rho_{min}}{1 + e^{slope - i - j}} \ ,$$

ρ_{min} is the minimal value assumed by ρ (i.e., the maximum time discount), and slope rules the position where the slope is steepest (Figure 6.2).

Some observations on the above definition:

- The use of the term $\sigma_{que}(s_i, s'_j) - \theta$ implies that query pairs whose similarity is above (below) θ are considered as matches (mismatches). Although a "sharp" threshold is used, the score of a matching pair and the cost of a mismatching pair turn out to be proportional to the distance of that pair similarity from θ.
- The definition given of the gap penalty δ is such that it guarantees a gap penalty to be payed if it enables a good match (i.e. a match higher than the average).
- The time-discounting function ρ leads match and mismatch scores to decay when moving backwards along the two sessions; it is maximum and equal to 1 for the ending queries of the two sessions.

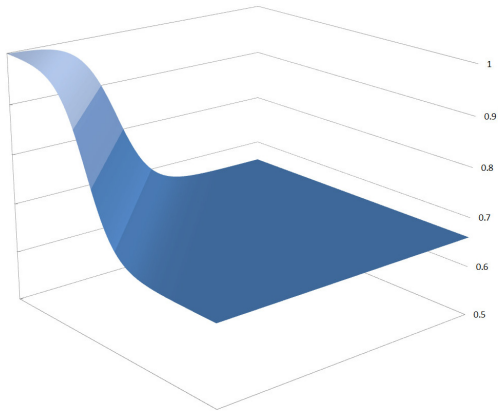

Figure 6.2: The time-discounting function $\rho(i, j)$ with $\rho_{min} = 0.66$ and $slope = 4$

	q_4	q_5	q_6	q_7	q_8
q_1	-0.004	0.171	0.120	-0.071	0.160
q_2	0.013	0.208	0.151	-0.053	0.186
q_3	-0.032	0.126	0.053	-0.082	0.132

Table 6.4: Query similarities for Example 6.4.5

The optimal alignment between s and s' is determined by the highest value in A, $\overline{A(s, s')}$, that we call *alignment score*. The positions $\bar{i}$ and $\bar{j}$ such that $A(\bar{i}, \bar{j}) = \overline{A(s, s')}$ mark the end of the matching subsequences of s and s'.

The alignment score is not really a similarity value, since it is not limited in the interval [0..1]. This creates problems when comparing sessions with difference length. Then we define OLAP session similarity by normalizing the alignment score:

Definition 6.4.5 (Alignment-Based Similarity of OLAP Sessions) *Let s and s' be two OLAP sessions on schema $\mathcal{M}$, of lengths v and v' respectively (with $v \leq v'$), and let $\overline{A(s, s')}$ be the alignment score for s and s'. The* alignment-based similarity *between s and s' is*

$$\sigma_{ali}(s, s') = \frac{\overline{A(s, s')}}{(1 - \theta) \sum_{k=1}^{v} \rho(k, k)}$$

where the normalizing factor is the alignment score for two identical sessions of length v.

Example 6.4.5 *Let s and s' be the two sessions to compare introduced in Example 6.4.2. Table 6.4 reports the query similarities, computed with $\alpha = \beta = \gamma = 1$ for simplicity, after the application of the time-discounting function ρ and using the threshold $\theta = 0.7$. Note that a negative value represents a mismatch, and a positive one a match. Table 6.5 shows the OLAP session alignment matrix for s and s'; the cells in bold denote alignments between two queries (e.g., q_1 is aligned with q_5), those in italics refer to gaps. Alignments on recent queries are favored, so q_3 is aligned with q_8. Query q_4 is not involved in the alignment due*

	q_4	q_5	q_6	q_7	q_8
q_1	0.000	**0.171**	0.120	0.000	0.160
q_2	0.013	0.208	**0.322**	*0.191*	0.186
q_3	0.000	0.139	0.261	0.241	**0.323**

Table 6.5: OLAP session alignment matrix for Example 6.4.5

to the low similarity it has with the other queries in s. In q_7, a gap penalty is paid to gain the good match between q_3 and q_8. The overall similarity between s and s' is 0.323 (the highest value in the matrix). After normalization, we obtain $\sigma_{ali}(s, s')$=0.387.

The properties of the proposed similarity function can be evaluated in terms of the distance function it induces using the standard transformation $\sigma_{ali} = 1/(1 + Dist_{ali})$. As stated in [22] for the original Smith-Waterman approach, $Dist_{ali}$ is not a metric because, while it is non-negative and symmetrical, it is not reflexive and it does not satisfy the triangular inequality as shown in Example 6.4.6. In particular, the triangular inequality cannot be satisfied because this approach is based on a local alignment.

Example 6.4.6 *Let $s = \langle q_1, q_2 \rangle$, $s' = \langle q_1, q_2, q_3, q_4 \rangle$, and $s'' = \langle q_3, q_4 \rangle$ be three sequences, where $\sigma_{que}(q_i, q_j) = 0$ if $i \neq j$. It is*

$$Dist_{ali}(s, s'') = \infty \ , \ Dist_{ali}(s, s') = Dist_{ali}(s', s'') = 0$$

which obviously contradicts the triangle inequality axiom. Besides, s' has zero distance from both s and s'' though $s \neq s' \neq s''$.

Note that, despite its complexity and the fact that it is not a metric, this extension of sequence alignment not only fulfills the requirements expressed in Section 6.2, but is close to the users' perception, without sacrificing efficiency of computation, as noted in the conclusion below. Moreover, it can be seen as a more demanding measure, as illustrated by the different similarity scores computed for the sessions of Example 6.4.2 (e.g., 0.96 for the extension of Tf-Idf against 0.387 for the extension of sequence alignment).

6.5 Conclusion

This chapter introduced various ways of measuring the similarity between queries and sessions. These similarity measures between sessions have been extensively tested in [10]. We briefly report the main results of these tests.

Subjective tests using the questionnaires mentioned in Section 6.2 showed that the query similarity computed through σ_{que}, the extensions of Tf-Idf and sequence alignment are the closest to the ones perceived by users. Objective tests with synthetic sessions over the CENSUS schema showed that the extension of sequence alignment outperforms the other extensions as long as the order of queries in the sessions is considered relevant. As to efficiency, as expected, the Dice extension is the most efficient followed by the sequence alignment extension and Levenshtein extension, but these latter evolve less quickly due to their

linear programming nature. The Tf-Idf-based extension is the less efficient. Overall, the time required for comparing two sessions is perfectly compatible with complex applications.

The measures introduced here would be quite beneficial for different classes of applications, like for instance workload clustering [42] or optimization [93]. The following chapters, especially Chapter 9 on query recommendations, show how these measures can also be used for user-centric OLAP.

Part IV

Log-driven user-centric analysis

Chapter 7

Extracting profile information from the log

This chapter describes how relevant knowledge can be extracted from an OLAP query log to subsequently support OLAP analyses. In the case of a log of multidimensional queries, Section 7.1 describes how simple preferences over dimensions, members and measures can be discovered. Section 7.2 describes how navigational habits can be discovered, for instance under the form of correlations between query fragments. Finally, when query results can be analyzed, the measure values queried can be the basis for discovering what the user was looking for, which is presented in Section 7.3.

This chapter relies on material published in [16], [81], [9] and [39].

7.1 Extracting simple preferences over multidimensional data

If the log consists of OLAP queries from which fragments can be extracted, an approach inspired by that of Holland & al. [53] can be used to extract simple preferences over dimensions, members and measures. We first start by describing the preferences over multidimensional data, and then present the algorithm for extracting these preferences from a log of OLAP queries.

7.1.1 Preference definition

The preferences defined here are simple preferences over dimensions, members and measures, that allow for instance to order multidimensional queries. More complex preferences over multidimensional data can be found in [46], e.g., to handle preferences over levels, and in [81], to take visualization structures into account.

We define simple preferences over multidimensional data to be: a partial order over the dimensions, dimension-wise partial orders over members, and

an order over measures. Formally, given a cube $C = \langle D_1, ..., D_n, F\rangle$ over a multidimensional schema $\mathcal{M} = \langle A, H, M\rangle$, such preferences are $\mathcal{U} = \langle <^u_{\mathcal{D}}, \{<^u_{D_1}, \ldots, <^u_{D_n}, <^u_{Meas}\}\rangle$ where:

- $<^u_{\mathcal{D}}$ is a partial order over dimensions $\mathcal{D} = \{D_1, \ldots, D_n\}$ of the cube. Given two dimensions D_i and D_j of $\mathcal{D}$, D_j is preferred to D_i if $D_i <^u_{\mathcal{D}} D_j$.
- for all $i \in [1, n]$, $<^u_{D_i}$ is a partial order over the set of members in each dimensions $D_i \in \mathcal{D}$ i.e., a partial order over $\pi^*(D_i)$, $i \in [1, n]$. Given two members m and m' in D_i, m is less preferred than m' if $m <^u_{D_i} m'$.
- $<^u_{Meas}$ is a partial order over the measure set M. Given two measures $meas$ and $meas'$ of M, $meas$ is preferred to $meas'$ if $meas' <^u_{Meas} meas$. dimensions

Example 7.1.1 *An example of simple preferences over the* CENSUS *schema is:*

$\langle$
$\{RESIDENCE <^u_{\mathcal{D}} TIME <^u_{\mathcal{D}} SEX,$
$RESIDENCE <^u_{\mathcal{D}} TIME <^u_{\mathcal{D}} OCCUPATION,$
$RESIDENCE <^u_{\mathcal{D}} RACE <^u_{\mathcal{D}} SEX,$
$RESIDENCE <^u_{\mathcal{D}} RACE <^u_{\mathcal{D}} OCCUPATION\},$
$\{Male <^u_{SEX} Female\},$
$\{AvgIncome, AvgCostWatr <^u_{Meas} AvgCostGas <^u_{Meas} AvgCostElec\}$
$\rangle$.

7.1.2 Preference extraction

Obviously, an order over references can straightforwardly be derived from a log where queries are modeled as sets of references, by counting the occurrences of the references in the log. In what follows, we consider that logged queries are not evaluated, and modeled as sets of fragments (qf-set).

In [53], orders on constants are derived from the frequency of occurrences of the constants in the log. We will follow this idea and adapt it to our model of preferences. This model relies on an order over dimensions, dimension-wise orders over members, and an order over measures. This is handled by considering that the order over dimensions is inferred from the former queries' projections, over levels or measures, and the orders over members are inferred from the former queries' selections.

The order over dimensions $<^u_{\mathcal{D}}$ is thus constructed as follows:

a) for each dimension $D = \{L_0, \ldots, L_d\}$, let $count(D)$ be the number of times one of its levels (different from L_d, i.e., the coarsest level) appears in a group-by set of the queries in the log.

b) Order the dimensions accordingly, i.e., for each D_i, D_j, $D_i <^u_{\mathcal{D}} D_j$, if $count(D_i) < count(D_j)$, and considering that if $count(D_i) = count(D_j)$ then $D_i \sim^u_{\mathcal{D}} D_j$.

It can easily be seen that the resulting order is a weak order. Note that an order over levels in a dimension can be easily computed using this principle.

For each dimension D, the order $<^u_D$ is constructed as follows:

a) for each value v in $\pi^*(D)$, let $count(v)$ be the number of times it appears in a selection predicate of the queries in the log.

b) Order the values accordingly, considering that if $count(v_i) = count(v_j)$ then $v_i \sim^u_D v_j$

Finally, $<^u_{Meas}$ is constructed as follows:

a) for each measure $meas$, let $count(meas)$ be the number of times it appears in the log.

b) Order the values accordingly, considering that if $count(meas) = count(meas')$ then $meas \sim^u_{Meas} meas'$

It can easily be seen that the resulting orders are weak orders.

Example 7.1.2 *Consider the log $L = \{s_1, s_2\}$ with $s_1 = \langle q_1, q_2 \rangle$, $s_2 = \langle q_3 \rangle$ and queries, over the* CENSUS *schema, are modeled as the following query fragment sets:*
$q_1 =$
$\{AllCities, Sex, AllYears, AllRaces, Occs, (Sex = Female), AvgCostElec\}$,
$q_2 =$
$\{AllCities, Sex, Year, AllRaces, Occs, (Sex = Female), AvgCostElec\}$,
$q_3 =$
$\{AllCities, Sex, AllYears, RaceGroup, Occs, (Sex = Male), AvgCostGas\}$.

The preference extraction principle introduced above can be used to extract from this log the simple preferences of Example 7.1.1. For instance, since predicate $(Sex = Female)$ appears twice in the log while predicate $(Sex = Male)$ appears only once, it is $Male <^u_{SEX} Female$. Similarly, it is $RESIDENCE <^u_{\mathcal{D}} TIME <^u_{\mathcal{D}} SEX$ since level Sex appears three times, level $Year$ appears only once, and no level other than the coarsest level is used for dimension RESIDENCE.

Note that in this approach, counting is irrespective of sessions. It can be straightforwardly extended by counting not the number of occurrences of each fragment in the log, but the number of sessions where the fragment appears.

7.2 Extracting navigational habits

The log can be searched for patterns to characterize the navigations it contains. Referring to such patterns, we use the term *navigational habit*, to denote a pattern present in the query log, that makes explicit a behavior of the user. Obviously, many types of patterns can be extracted from a log. We give here a basic approach for extracting simple patterns, under the form of association

Algorithm 3 Extract rules with support and confidence adjustment

Input: Log: A set of queries; $minSup, minConf$: Floats
Output: R: A set of association rules
Uses: $mine(set, float, float)$: An association rule extractor
Variables: $stop$: A Boolean; $confidence, support$: Floats; $Covered$: A set of qf-sets

```
1:  stop =false
2:  confidence = 1
3:  support = 1
4:  while !stop do
5:      R = mine(Log, support, confidence)        ▷ Mine rules above support and confidence
6:      Covered = ∅
7:      for each rule r ∈ R do
8:          Covered = Covered ∪ {q ∈ Log | r.ant ∪ r.cons ⊆ q}
9:      if Covered = Log then                     ▷ If all queries in the log are covered in R stop...
10:         stop =true
11:     else                                      ▷ ...else mine again with lower thresholds
12:         confidence = confidence − 0.1
13:         if confidence < minConf then
14:             support = support − 0.1
15:             confidence = 1
16:             if support < minSupp then
17:                 stop = true
18: return R
```

rules, from a log of unevaluated queries. This approach can be seen as a starting point for extracting more sophisticated types of patterns (sequence patterns, etc.).

7.2.1 Simple navigational habits

In the approach we present, the log consists of OLAP queries from which fragments can be extracted. A navigational habit is defined by an association rule of the form $X \rightarrow Y$, with the intuitive meaning that whenever fragment X is used in a query expression, then fragment Y tends to be present as well.

Definition 7.2.1 (Simple navigational habit) *A simple navigational habit is a rule of the form $X \rightarrow Y, (s, c)$, where X and Y are query fragments and s and c, for support and confidence, are real numbers.*

Example 7.2.1 *The habit $Year \rightarrow Region(0.7, 0.8)$ indicates that queries involving $Year$ in the group-by set involve also $Region$ in 80% of the cases, the pair $Year, Region$ appearing in 70% of the queries of the log.*

7.2.2 Extracting habits

The input of the algorithm for extracting navigational habits is a set of qf-sets that represents the user's query log, while the output is a set R of association rules. We recall that a qf-set is a set of query fragments, i.e., a set of levels, measures or selection predicates.

Interestingly, the problem of associating a query with a set of fragments representing user preferences bears resemblance to the problem of associating objects with a set of most relevant labels. This problem, named *label ranking*, is a form of classification. Both label ranking and classification have been proved

to be effectively handled by association rules (see for instance [31, 68]). In this context, rules have as antecedent a set of features that should match the object to be classified, and one label as consequent. We adopt a similar approach here, and we search for rules having exactly one item as consequent, so each rule $r \in R$ takes the form $ant \rightarrow cons$, where ant is a qf-set and $cons$ is a single query fragment. In the following, $r.cons$ (resp., $r.ant$) denotes the consequent (resp., antecedent) of rule r, and $conf(r)$ its confidence.

The algorithm also uses any classical association rule extractor that is parametrized by support and confidence thresholds (e.g., Apriori [7]). The only issue here is to extract rules that faithfully represent the user's query log. Since the user is not involved, support and confidence have to be adjusted automatically [31]. Algorithm 3 is used for this purpose, and it extracts rules until the whole log is covered by the set of rules extracted, in the spirit of what is done for class association rule extraction (see e.g., [68]). More precisely, the algorithm starts extracting rules with confidence and support equal to 1 (lines 2,3). If the set of rules covers the entire log, then the algorithm stops (line 9, 10). Otherwise, extraction starts again with a lower confidence (line 14), and confidence is decreased until the log is entirely covered or the confidence is considered too low (line 13). In this case, confidence goes back to 1 and support is decreased (line 14,15), and extraction is launched again. If both support and confidence are considered too low, then the algorithm stops.

Algorithm 3 needs two thresholds, $minConf$ and $minSupp$. Realistic values for these thresholds can be learned by training the algorithm on query logs, or be derived from log properties like size and sparseness.

Example 7.2.2 *Consider the log given in Example 7.1.2. The habits will be extracted from the log with support and confidence equal to 1. In particular,* $AllCities \rightarrow Sex, (1, 1)$ *is such a habit.*

7.3 Extracting analysis discoveries

If the query expressions in the log are grouped into sessions and are modeled as unevaluated queries (i.e., represented by their results), the log can be processed to discover what was the goal of the user during the analysis session. In this section, we describe a technique for detecting what the user was investigating in the context of discovery driven analysis [94, 95, 97], by mining a log of fully evaluated queries.

Discovery driven analysis of OLAP cubes was introduced by Sarawagi et al. [96] to support interactive analysis of multidimensional data. It consisted in the definition of advanced OLAP operators to guide the user towards unexpected data in the cube or to propose to explain an unexpected result. In particular, The DIFF operator proposed in [94] explores the reasons why an aggregate is lower (or higher) in one cell compared to another. The RELAX operator [97] tries to confirm at a lesser level of detail a particular significant difference, and summarizes the exceptions to this difference.

The key idea of processing the log to discover what was the goal of the user, is

to detect the difference between measure values that queries investigated. More precisely, the log is examined to discover the pairs of cells whose measures differ significantly, to retain the most general ones (the most general difference pairs, *mgdp*) as well as the queries that contains them (the most general difference queries, *mgdq*). For such pairs, a structure called investigation is created that records the set of *mgdq* and, at a lower level of detail, the queries that confirm the difference (their drill-down differences), and the queries that contradicts the difference (their exceptions).

Example 7.3.1 *Consider the following set of query answers, corresponding to a log L of 3 sessions* $s_1 = \langle q_1, q_2 \rangle, s_2 = \langle q_3, q_4 \rangle, s_3 = \langle q_5, q_6, q_7 \rangle$*, all investigating the average income according to sex, residence and time.*

s_1

q_1		L.A.	N.Y.
2007	Female	200	100
	Male	200	250

q_2		L.A.	N.Y.
2008	Female	100	200
	Male	200	250

s_2

q_3		Pacific	Atlantic
2007	Female	150	150
	Male	200	250

q_4		Pacific	Atlantic
2008	Female	100	200
	Male	200	250

s_3

q_5	Pacific	Atlantic
Female	100	200
Male	200	200

q_6	California	New York
Female	150	150
Male	200	250

q_7	L.A.	N.Y.
Female	150	150
Male	200	250

7.3.1 Identifying relevant pairs of cells

We start by defining relations over pairs of cells and over queries. First note that the specialization relation over cells can be extended to pairs of cells in the following way.

Definition 7.3.1 (Specialization over pairs) *Let C be a cube and c, c', c'', c''' be four cells of C. The pair $\langle c, c' \rangle$ is a generalization of $\langle c'', c''' \rangle$, noted $\langle c, c' \rangle \succeq_c \langle c'', c''' \rangle$ if both $c \succeq_c c''$ and $c' \succeq_c c'''$.*

If we have $\langle c, c' \rangle \succeq_c \langle c'', c''' \rangle$, we will say that $\langle c, c' \rangle$ is a *rollup pair* of $\langle c'', c''' \rangle$ and $\langle c'', c''' \rangle$ is a *drilldown pair* of $\langle c, c' \rangle$. Moreover, if $\langle c'', c''' \rangle$ is a drill-down pair of $\langle c, c' \rangle$ and $sign(measure(c'') - measure(c''')) \neq sign(measure(c) - measure(c'))$ we will say that $\langle c'', c''' \rangle$ is an *exception pair* of $\langle c, c' \rangle$. Given a set S of pairs of cells, the most general pairs are the pairs of S that have no rollup pairs in S.

Definition 7.3.2 (Most general pairs) *Let S be a set of pairs of cells. The most general pairs of S are the set $max_{\succeq_c}(S)$. For a given pair of cells $\langle c, c' \rangle$ of S, the most general pairs for $\langle c, c' \rangle$ in S is the set $max_{\succeq_c}(\{\langle c'', c''' \rangle \in S | \langle c'', c''' \rangle$ is a rollup pair of $\langle c, c' \rangle\})$.*

Example 7.3.2 *Consider the query answers of Example 7.3.1. For the sake of readability, the cells are given using only the three hierarchies utilized. The pair:*

$$\langle\langle 2007, Female, L.A., 200\rangle, \langle 2007, Female, N.Y., 100\rangle\rangle$$

of query q_1 is a drill-down pair of the pair:

$$\langle\langle 2007, Female, Pacific, 150\rangle, \langle 2007, Female, Atlantic, 150\rangle\rangle$$

of query q_3. A most general pair of this log is:

$$\langle\langle AllYears, Female, Pacific, 100\rangle, \langle AllYears, Female, Atlantic, 200\rangle\rangle$$

of query q_5.

In what follows we will call a significant difference (or difference for short) a pair of cells such that their measures differ significantly. This significance is computed in two steps. First, a user-defined function fdp on which we do not impose particular requirements, is used to detect base difference pair.

Definition 7.3.3 (Base difference pair) *Let C be a cube, fdp be a boolean function over the pairs of cells of C and c', c be two cells of C. The pair $\langle c, c'\rangle$ is a difference pair for C and fdp if $fdp(c, c') = true$.*

A first operator, *difference*, outputs the pairs of cells of a query q that are difference pairs, i.e., $difference(fdp, q) = \{\langle c, c'\rangle \in q | fdp(c, c') \text{ is } true\}$ for some boolean function fdp over pairs of cells.

For a base difference pair $\langle c, c'\rangle$, we define its roll-up (resp., drill-down) difference pairs as its roll-up (resp., drill-down) pairs that show a significant difference w.r.t. $\langle c, c'\rangle$ in the sense of a given function.

Definition 7.3.4 (Roll-up/drill-down difference pair) *Let $\langle c, c'\rangle$ be a difference pair, $\langle c'', c'''\rangle$ be one of its roll-up (resp., drill-down) pairs and r be Boolean function over couples of pairs of cells. We say that $\langle c'', c'''\rangle$ is a roll-up difference pair (resp., $\langle c, c'\rangle$ is a drill-down difference pair) for $\langle c, c'\rangle$ if $r(c, c', c'', c''') = true$.*

The next operators detect, for a pair $\langle c, c'\rangle$, a set of queries Q and a Boolean function r, which are the pairs of Q that are roll-up (resp., drill-down, resp., exception) difference pairs for$\langle c, c'\rangle$. Formally,

- $rollupDifferencePairs(c, c', Q, r) = \{\langle c'', c'''\rangle | q \in Q$ with $\langle c'', c'''\rangle \in q$ and $\langle c, c'\rangle \succeq_c \langle c'', c'''\rangle$ and $r(c, c', c'', c''') = true\}$,
- $drilldownDifferencePairs(c, c', Q, r) = \{\langle c'', c'''\rangle | q \in Q$ with $\langle c'', c'''\rangle \in q$ and $\langle c'', c'''\rangle \succeq_c \langle c, c'\rangle$ and $r(c, c', c'', c''') = true\}$,
- $exceptionPairs(c, c', Q) = \{\langle c'', c'''\rangle | q \in Q$ with $\langle c'', c'''\rangle \in q$ being an exception pair for $\langle c, c'\rangle\}$.

In what follows, base difference pairs, roll-up difference pairs and drill-down difference pairs will be called simply difference pairs.

Example 7.3.3 *Consider the query answers of Example 7.3.1, and two functions: (i) fdp that outputs true for two cells c, c' if their measures differ by at least a factor of 2 and (ii) r that outputs true for two pairs of cells $\langle c, c' \rangle$ and $\langle c'', c''' \rangle$ if $\frac{measure(c)}{measure(c')} \simeq \frac{measure(c'')}{measure(c''')}$. Then $p_1 = \langle \langle AllYears, Female, Pacific, 100 \rangle, \langle AllYears, Female, Atlantic, 200 \rangle \rangle$ is a base difference pair and $drilldownDifferencePairs(p_1, queries(L), r) =$ $\{\langle \langle 2008, Female, Pacific, 100 \rangle, \langle 2008, Female, Atlantic, 200 \rangle \rangle,$ $\langle \langle 2008, Female, L.A., 100 \rangle, \langle 2008, Female, N.Y., 200 \rangle \rangle\}$.*

The pair $\langle \langle 2007, Female, L.A., 200 \rangle, \langle 2007, Female, N.Y., 100 \rangle \rangle$ is an exception pair of p_1.

7.3.2 Identifying relevant queries

We define a difference query to be a query whose result displays one or more difference pairs. A query is a roll-up (resp., drill-down) difference query of a difference query if its result confirms the difference at a higher (resp., lower) level of detail. An exception is a query which result contradicts a difference at a lower level of detail. The following definitions formalize these notions.

Definition 7.3.5 (Difference query) *Let C be a cube, fdp be a boolean function over the pairs of cells of C. A query q over C is a difference query if there exists two cells $c, c' \in q$ such that the pair $\langle c, c' \rangle$ is a difference pair for C and fdp.*

Definition 7.3.6 (Rollup/drilldown/exception difference query) *Let q and q' be two queries and let $\langle c, c' \rangle$ be a difference pair in q. We say that q' is a rollup (resp., drill-down/exception) difference query for q if there exists a difference pair $\langle c'', c''' \rangle$ in q' that is a roll-up (resp., drill-down, resp., exception) difference pair of $\langle c, c' \rangle$. q' is said to be a roll-up (resp., drill-down, resp., exception) difference query for q w.r.t. the pair $\langle c, c' \rangle$.*

The next operators detect, for a pair $\langle c, c' \rangle$ and a set of queries Q, which are the queries of Q that are rollup (resp., drill-down/exception) difference queries w.r.t. $\langle c, c' \rangle$. Formally,

- $rollupDifference(c, c', Q) = \{q \in Q | q$ is a rollup difference query w.r.t. $\langle c, c' \rangle\}$,
- $drilldownDifference(c, c', Q) = \{q \in Q | q$ is a drilldown difference query w.r.t. $\langle c, c' \rangle\}$,
- $exception(c, c', Q) = \{q \in Q | q$ is and exception difference query w.r.t. $\langle c, c' \rangle\}$

A most general difference query ($mgdq$) is a query that contains a most general difference pair.

Algorithm 4 Detecting investigations

Input: A log L, a boolean function fdp
Output: A set I of investigations
Variables: sets I, DP, RDP, MP

```
1: I, DP, RDP, MP ← ∅
2: for each query q queries(L) do                          ▷ detect base difference pairs
3:     DP = DP ∪ difference(fdp, q)
4: for each pair p ∈ DP do                  ▷ detect all their roll-up difference pairs
5:     RDP = RDP ∪ rollupDifferencePairs(p, queries(L), r)
6: MDP = max_{≥_c}(RDP)                                    ▷ retain only the most general
7: for each m ∈ MDP do
8:     M = {q ∈ queries(L) | m ∈ q}
9:     D = drilldownDifference(m, queries(L))              ▷ detect drill-down difference queries
10:    E = exception(m, queries(L))                                ▷ detect exception queries
11:    if D ≠ ∅ or E ≠ ∅ then                              ▷ update the set of investigations
12:        I = I ∪ < m, M, D, E >
13: return I
```

Definition 7.3.7 (mgdq) *Let q be a query, S be a set of pairs of cells and $\langle c, c' \rangle$ be a pair in S. q is a mgdq if it contains a pair of S that is a most general difference pair of S.*

Example 7.3.4 *Consider again the answers of Example 7.3.1. q_5 is a difference query since it contains the difference pair p_1. It is also a mgdq of the queries in L. q_2 and q_4 are two drill-down difference queries for q_5. q_1 is an exception query for q_5.*

7.3.3 Extracting investigations

Algorithm 4 processes each session to discover the mgdq, their drill-down differences and exceptions. This algorithm outputs a set of what we call *investigations*, i.e., the various queries that investigated a particular difference pair. Note that one investigation is created per mgdq discovered in the log, provided this mgdq comes with some drill-down difference pairs or exception pairs.

Definition 7.3.8 (Investigation) *An investigation i for a log L is a tuple $\langle \langle c, c' \rangle, M, D, E \rangle$ where $\langle c, c' \rangle$ is a most general difference pair appearing in L, M, D and E are subsets of queries(L), M is the set of queries that contains $\langle c, c' \rangle$, D is the set of drill-down difference queries w.r.t. $\langle c, c' \rangle$, E is the set of exception difference queries w.r.t. $\langle c, c' \rangle$, and at-least one of D, E is non-empty.*

Note that a query can be at the same time mgdq or drilldown difference or exception. Thus the queries in i are labelled with their type (mgdq, drill-down difference or exception) and are associated with their pair of cells that is the drill-down or exception pair w.r.t. the mgdq. In an investigation $i = \langle m, M, D, E \rangle$ m is called the difference pair of the investigation. For a set I of investigations, mgdq (I) is the set of mgdq of every investigation in I.

Example 7.3.5 *Following the previous examples, starting from Example 7.3.1, an investigation that can be extracted from the query answers is: $i_1 = \langle p_1, \{q_5\}, \{q_2, q_4\}, \{q_1\} \rangle$. This investigation indicates that (the result of) query*

q_5 contains pair p_1 that shows a high difference in the average income of Females between Pacific and Atlantic regions. Such a difference can also be observed in (the result of) queries q_2 and q_4, at a finer granularity level, while (the result of) query q_1 contradicts this observation. Another investigation for the difference pair $p_2 = \langle\langle AllYears, Female, Pacific, 100\rangle, \langle AllYears, Male, Pacific, 200\rangle\rangle$ is $i_2 = \langle p_2, \{q_5\}, \{q_2, q_4\}, \emptyset\rangle$. This investigation indicates that query q_5 also shows a high difference in the Pacific region between the average income of Females and the average income of Males. Such a difference can also be observed in queries q_2 and q_4, at a finer granularity level, and no contradiction to this observation is to be found in the other queries of the log.

7.4 Conclusion

This chapter introduced various knowledge hidden in a query log, together with techniques for extracting them. Extracted knowledge include (i) simple preferences over dimensions, members and measures, (ii) correlations between query fragments, and (iii) investigations representing what has been seen during past sessions for a pair of cells where measures differ significantly. Unsurprisingly, it can be seen that, the more precise the query model used (with respect to the database instance), the more complex the information extracted is.

The next two chapters describe how this knowledge can be leveraged for user-centric OLAP, i.e., supporting cube analysis, by introducing personalization and recommendation approaches that rely on the knowledge discovered. These approaches will be presented regarding the query model used, that, as stated in Chapter 3, reveals if the focus is to be attached to the way the query is written (unevaluated queries), or the part of the cube queried (partially evaluated queries), or the values retrieved (fully evaluated queries).

Chapter 8

Personalizing queries with a single user log

In this chapter, we present two approaches for personalizing multidimensional queries, one based on the use of dedicated operators and one based on query expansion. We essentially view query personalization as a way to deal with the potentially large answer set returned by evaluating the query. This is of particular importance in the context of OLAP since the databases queried are often very large, user may not know precisely what part of the database to explore, and the device used for visualizing the answer may not be appropriate for visualizing large cross-tabs, which is the classical form for multidimensional answers. Thereby, Section 8.1 presents approaches for deducing from the log the preference constructs to be used with a user query, when a preference language can be used. Section 8.2 presents prescriptive approaches for expanding the user's current query, using preferences extracted from the log.

Our work on personalization started in the context Hassina Mouloudi's PhD thesis [81], whose most salient publication is [16]. This chapter relies on this material, as well as on material published in [9].

8.1 Use of dedicated operators

Using a log allows to perform personalization with a non prescriptive approach requiring low formulation effort. In what follows, the profile consists of simple preferences or navigational habits extracted from the log (see Chapter 7).

8.1.1 The MyMDX preference language

The language we adopt in this section to express OLAP preferences is MYMDX [19], an extension of the MDX language based on the MYOLAP algebra [46]. In this section we summarize its features of interest for this work.

A (qualitative) preference on a data cube is a *strict partial order* (i.e., an

irreflexive and transitive binary relation) on the space $\mathcal{F}_{\mathcal{M}}$ of all facts. In the MYOLAP algebra, preferences are inductively engineered by writing a *preference expression* that can be either a *base constructor* or a *composition operator* applied to two preference expressions. The constructors used here are[1]:

- POS(a, V), where a is a level and $V \subset Dom(a)$, that operates on level values; facts for which a takes a value in V are preferred to the others.
- BETWEEN(m, v_{low}, v_{high}), where m is a measure and $v_{low}, v_{high} \in Dom(m)$, that operates on measure values. Facts whose value of m is between v_{low} and v_{high} are preferred; the other facts are ranked according to their distance from the $[v_{low}, v_{high}]$ interval.
- CONTAIN(h, L), where h is a hierarchy and $L \subset Lev(h)$, that operates on levels. Facts whose group-by set includes a level in L are preferred to the others.
- CONTAIN(measures, $Meas$), where $Meas \subset M$, that operates on measures. Facts whose measure is in $Meas$ are preferred to the others.

Preference composition relies on the Pareto operator ($\otimes$), that gives the same importance to both the composed preferences. Remarkably, the Pareto operator is closed on the set of preferences.

The MYMDX language allows an MDX query to be annotated with a preference expression through a PREFERRING clause.

Example 8.1.1 *We recall the MDX query on the* CENSUS *schema of Example 3.2.1:*

```
SELECT AvgIncome ON COLUMNS,
        Crossjoin(OCCUPATION.members,
                Crossjoin(Descendants(RACE.AllRaces,RACE.Mrn),
                Descendants(RESIDENCE.AllCities,RESIDENCE.Region))) ON ROWS
FROM CENSUS WHERE TIME.Year.[2009]
```

This query can be annotated with preference expression:
BETWEEN(AvgIncome,500,1000) $\otimes$ POS(Occ,'Engineer') $\otimes$ CONTAIN (RESIDENCE, Region)
to state that facts aggregated by region and related to engineers with average income between 500 and 1000 kilo-euros are equally preferred. The corresponding MYMDX *query is:*

```
SELECT AvgIncome ON COLUMNS,
        Crossjoin(OCCUPATION.members,
                Crossjoin(Descendants(RACE.AllRaces,RACE.Mrn),
                Descendants(RESIDENCE.AllCities,RESIDENCE.Region))) ON ROWS
```

[1]The constructors we adopt are actually a generalization of those presented in [46] from two points of view. Firstly, the CONTAIN constructor is extended to work also on a fake hierarchy including all measures. Secondly, all constructors except BETWEEN are extended to operate on sets of values rather than on single values.

FROM CENSUS **WHERE** TIME.Year.[2009]
PREFERRING AvgIncome **BETWEEN** 500 **AND** 1000
AND Occ **POS** 'Engineer' **AND** RESIDENCE **CONTAIN** Region

8.1.2 Using unevaluated queries

The approach for personalizing a user query with MYMDX, using extracted navigational habits of the form of rules introduced in Section 7.2.2, relies on the following three steps:

a) *Rule selection.* When a user formulates an MDX query q, a subset $R_q \subseteq R$ of rules is selected. Each rule in R_q is *pertinent* w.r.t. q, meaning that its antecedent matches with q, and *effective*, meaning that the preference it would be translated into can actually induce an ordering on the facts returned by q. Then, let a positive integer *personalization degree* α be chosen by the user to express the desired preference complexity. A qf-set F_α is generated from R_q in such a way that the final preference expression of the MYMDX statement has α base constructors, where α is user-defined.

b) *Fragment translation.* Each fragment in F_α is translated into a base constructor; the resulting base constructors are then coalesced and composed using the Pareto operator into a preference expression p.

c) *Querying.* Query q is annotated with p, translated into MYMDX, and executed. As shown in [19], the user can effectively explore query results by visually interacting with a graph-like structure that emphasizes the better-than relationships induced by p between different sets of facts. Preferred facts are then displayed in a multidimensional table.

The following subsections explain in detail how steps 1, 2, and 3 are carried out. We recall that, $r.cons$ (resp., $r.ant$) denotes the consequent (resp., antecedent) of rule r, and $conf(r)$ its confidence.

Rule Selection

The navigational habits R may be a large set of rules. In this section we present the algorithm that first selects, among the rules in R, the subset R_q of pertinent and effective rules for query q, and then returns a qf-set F_α including a subset of the query fragments that appear as consequents of the rules in R_q. These fragments will be used for annotating q with a preference.

Following the approach presented in [108], the selection of query fragments is made by associating a score to each group of rules in R_q having the same fragment φ as consequent. This score is the average confidence of the rules in the group, i.e., $score(\varphi) = avg_{r \in R_\varphi} conf(r)$ where $R_\varphi \subseteq R_q$ is the subset of rules having φ as a consequent. The selected query fragments are those with highest scores, and are limited by the number α of base preference constructors that the user wants to annotate her queries with.

Given schema $\mathcal{M} = \langle A, H, M \rangle$ and a qf-set F, we adopt the following notation:

- $F.hier(h) = F \cap Lev(h)$ is the set of levels of hierarchy $h \in H$ in F;
- $F.meas = F \cap M$ is the set of measures in F;
- $F.val(a) = \bigcup_{(a \in V_k) \in F} V_k$ denotes the set of selected values for level/measure $a \in A \cup M$ in F.

Algorithm 5 Select fragments for personalisation

Input: R: A set of rules; q: A query represented as a qf-set; α: A user-defined *personalization degree*
Output: F_α: A qf-set that will be used to annotate q with a preference
Variables: $numBC$: The current number of base constructs; R_q: The set of pertinent and effective rules; F, F_{sim}: Two qf-sets

1: $R = R \setminus \{r \in R | r.ant \not\subseteq q\}$ ▷ Drop non-pertinent rules
2: $R_q = R \setminus \{r \in R | r.cons \in A \cup M, r.cons \notin q\}$ ▷ Drop non-effective rules
3: $F = \{r.cons | r \in R_q\}$ ▷ Consequents of the rules in R_q
4: $F_\alpha = \emptyset$
5: $numBC = 0$
6: **while** $numBC \leq \alpha$ and $F \neq \emptyset$ **do** ▷ Iteratively construct F_α...
7: let $\varphi = argmax_F\ score(\varphi)$ ▷ ...starting with the fragment having highest score
8: $F = F \setminus \{\varphi\}$
9: **if** $makesIneffective(\varphi, F_\alpha, q)$ **then** ▷ If φ drives the preference ineffective...
10: $F_{sim} = \{\varphi' \in F_\alpha | similar(\varphi, \varphi')\}$ ▷ ...find the similar fragments, if any...
11: $F_\alpha = F_\alpha \setminus F_{sim}$ ▷ ...and drop them
12: **if** $F_{sim} \neq \emptyset$ **then**
13: $numBC - -$
14: **else**
15: **if** $\exists \varphi' \in F_\alpha | similar(\varphi, \varphi')$ **then** ▷ Similar fragments were already added to F_α...
16: $F_\alpha = F_\alpha \cup \{\varphi\}$ ▷ ...so $numBC$ must not be increased
17: **else**
18: **if** $numBC < \alpha$ **then** ▷ Add φ only if this does not violate the α constraint
19: $F_\alpha = F_\alpha \cup \{\varphi\}$
20: $numBC + +$
21: **return** F_α

Function 6 makesIneffective

Input: φ: A fragment; F_α: A qf-set; q: a query represented as a qf-set
Output: A Boolean

1: **if** $\exists h \in H | \varphi \in Lev(h)$ **then** ▷ φ is a level
2: **if** $(F_\alpha.hier(h) \cup \{\varphi\}) = q.hier(h)$ **then** ▷ All query hierarchies are preferred
3: return true
4: **if** $\varphi \in M$ **then** ▷ φ is a measure
5: **if** $(F_\alpha.meas \cup \{\varphi\}) = q.meas$ **then** ▷ All query measures are preferred
6: return true
7: **if** $\varphi = (a \in V)$ **then** ▷ φ is a predicate
8: **if** $q.val(a) \neq \emptyset$ and $!((F_\alpha.val(a) \cup V) \subset q.val(a))$ **then** ▷ All values for a are preferred
9: return true
10: **return** false

Algorithm 5 selects, among the set R of association rules mined from the log, the consequents of rules that will be used to annotate the current query with preferences. It starts by removing from R all non-pertinent rules (i.e., those whose antecedent does not match q — line 1), and some non-effective rules (those whose consequent, if it is an attribute or a measure, does not appear in the list of group-by attributes or returned measures of q — line 2). The remaining rules are grouped by their consequent and the score of each group is computed (line 3). Then the top consequents corresponding to α base constructors are returned

Function 7 similar

Input: φ_1: A fragment; φ_2: A fragment
Output: A Boolean
1: **if** $\exists h \in H | \varphi_1 \in Lev(h)$ and $\varphi_2 \in Lev(h)$ **then** ▷ Two levels of the same hierarchy
2: return true
3: **if** $\varphi_1 \in M$ and $\varphi_2 \in M$ **then** ▷ Two measures
4: return true
5: **if** $\varphi_1 = (a \in V_1)$ and $\varphi_2 = (a \in V_2)$ **then** ▷ Two predicates on the same attribute
6: return true
7: **return** false

(lines 4-21). If a fragment φ that is about to be selected drives the preferences ineffective because it states that *all* the query results are preferred (Function 6), it is removed together with the other similar fragments (lines 10-13).

Example 8.1.2 *Consider the qf-set of Example 3.2.1,* $q = \{\mathsf{Region}, \mathsf{AllCities}, \mathsf{Mrn}, \mathsf{AllRaces}, \mathsf{Occ}, \mathsf{Year}, \mathsf{AllSexes}, \mathsf{AvgIncome}, (\mathsf{Year} = 2009)\}$. *Let the set R of rules extracted from the log be as follows:*

r_1:	$(\mathsf{Region} \in \{\text{'Pacific','Atlantic'}\}) \rightarrow \mathsf{Year}$	*(0.8)*
r_2:	$\mathsf{Year} \rightarrow \mathsf{Region}$	*(0.80)*
r_3:	$\mathsf{Year} \rightarrow \mathsf{AllCities}$	*(0.60)*
r_4:	$\mathsf{AvgIncome} \rightarrow \mathsf{Region}$	*(0.60)*
r_5:	$\mathsf{Year} \rightarrow \mathsf{Sex}$	*(0.90)*
r_6:	$(\mathsf{Year} = 2009) \rightarrow \mathsf{Region}$	*(0.70)*
r_7:	$\mathsf{Year} \rightarrow (\mathsf{Year} = 2009)$	*(0.50)*
r_8:	$\mathsf{Year} \rightarrow (\mathsf{AvgIncome} \in [500, 1000])$	*(0.55)*
r_9:	$\mathsf{AvgIncome} \rightarrow \mathsf{Mrn}$	*(0.45)*
r_{10}:	$\mathsf{Occ} \rightarrow \mathsf{Region}$	*(0.70)*
r_{11}:	$\mathsf{Occ} \rightarrow \mathsf{Year}$	*(0.10)*
r_{12}:	$\mathsf{AvgIncome} \rightarrow \mathsf{Year}$	*(0.70)*

and let Algorithm 5 be called with $\alpha = 2$. First, the algorithm removes r_1 (non pertinent since its antecedent is not found in q) and r_5 (non effective since its consequent is not found in q). Then the remaining rules are grouped by their consequents, resulting in the set of fragments $F = \{\mathsf{Region}, \mathsf{AllCities}, (\mathsf{AvgIncome} \in [500, 1000]), (\mathsf{Year} = 2009), \mathsf{Mrn}, \mathsf{Year}\}$ *(listed by decreasing order of score). The fragments in F are now orderly explored. The first two fragments are not selected since, together, they drive the preference ineffective (they are exactly the fragments of hierarchy* RESIDENCE *included in q). Fragment* $(\mathsf{AvgIncome} \in [500, 1000])$ *is selected. Fragment* $(\mathsf{Year} = 2009)$ *is not selected since it corresponds precisely to the selection on* Year *of q. Then fragment* Mrn *is selected and, finally, Algorithm 5 outputs* $F_\alpha = \{(\mathsf{AvgIncome} \in [500, 1000]), \mathsf{Mrn}\}$.

Deriving preference constructors

The output F_α of Algorithm 5 is a qf-set used to annotate the current query q with a preference. To this end, each query fragment $\varphi \in F_\alpha$ is translated into a base constructor ; the resulting base constructors are then coalesced and composed using the Pareto operator.

The rules for translating fragment φ are explained below:

- if φ is a level $a \in A$, it is translated into a constructor CONTAIN(h, a), where h is the hierarchy a belongs to.
- If φ is a measure $m \in M$, it is translated into a constructor CONTAIN(measures,m).
- If φ is a Boolean predicate on a level, $(a \in V)$, it is translated into a constructor POS(a, V).
- If φ is a Boolean predicate on a measure, $(m \in [v_{low}, v_{high}])$, it is translated into a constructor BETWEEN(m, v_{low}, v_{high}).

The resulting base constructors are coalesced by merging all CONTAIN's on the same hierarchy, all POS's on the same level, and all BETWEEN's on the same measure.

Example 8.1.3 *The preference expression that translates the qf-set F_α in Example 8.1.2 is* $p =$ BETWEEN(AvgIncome,500,1000) $\otimes$ CONTAIN(RACE, Mrn). *The* MYMDX *formulation for q annotated with p is:*

```
SELECT AvgIncome ON COLUMNS,
        Crossjoin(OCCUPATION.members,
                Crossjoin(Descendants(RACE.AllRaces,RACE.Mrn),
                Descendants(RESIDENCE.AllCities,RESIDENCE.Region))) ON ROWS
FROM CENSUS WHERE TIME.Year.[2009]
PREFERRING AvgIncome BETWEEN 500 AND 1000 AND RACE CONTAIN Mrn
```

Note that, in this approach, it is assumed that navigational habits extraction is done off-line, for efficiency purpose. Should this efficiency constraint be relaxed, the approach could be improved by using the personalization degree α, that indicates how many preference constructs should be used for personalizing the query, to guide rule extraction. Roughly speaking, it means that Algorithm 3 (see Section 7.2) and Algorithm 5 should be merged, to extract, for a given support and confidence threshold, only pertinent and effective rules needed to ensure α constructs. Confidence and/or support are decreased progressively only if the number of constructs is not reached.

8.1.3 Using partially evaluated queries

When the user query is partially evaluated, i.e., corresponds to a set of references, simple preferences extracted from the log can be used to define an order over Q_{MDX}, the set of queries expressed in MDX over a given data cube. In that case, personalizing a user query q boils down to computing the preferred subquery of q, and using this subquery to derive the MYMDX preference constructs.

Finding the preferred subquery

Limiting the size of the answer is done using a constraint on the query. This constraint can for instance represent the properties of the device used for displaying the answer. We introduce the notion of *constraint* and *anti-monotone constraint*.

Definition 8.1.1 - Constraint. *A constraint v is a boolean function defined over Q_{MDX}. Given a query $q \in Q_{MDX}$, we say that q satisfies the constraint v if $v(q) = true$.*

A constraint v is anti-monotone *if for every MDX query q and q' in Q_{MDX}, if $q \sqsubseteq q'$ and $v(q') = true$, then we have $v(q) = true$.*

Example 8.1.4 *A simple example of constraint sets the maximum number n of references for the query. Formally, this constraint is defined by the function $v(q) = true \equiv |q| \leq n$, where q is a set of references.*

The user preferences $\mathcal{U} = \langle <^u_{\mathcal{D}}, \{<^u_{D_1}, \ldots, <^u_{D_N}, <^u_{Meas}\}\rangle$ induce a lexicographic order over references of the cube, noted $<^u_r$, which is defined for all references t and t' by: $t <^u_r t'$ if for all $D_k \in max_{<_{\mathcal{D}}}\Delta(t,t')$, we have $t(D_k) <_{D_k} t'(D_k)$ where $\Delta(t,t') = \{D_i \in \mathcal{D} \mid t(D_i) \neq t'(D_i)\}$. Note that, consistently with [81], this order is irrespective of $<^u_{Meas}$. However, this definition can be straightforwardly extended to include $<^u_{Meas}$ for defining a partial order on cell schemata (instead of cell references).

Preferences over partially evaluated queries, noted $<^u_R$, can be defined as a partial order over sets of references. Let $<^u_r$ be a partial order over a set E of references, $<^u_R$ is defined on 2^E by: for all subsets R_1 and R_2 of E, $R_1 <^u_R R_2$ if $R_1 \neq R_2$ and $(\forall t_1 \in R_1 \setminus R_2)(\exists t_2 \in R_2 \setminus R_1)(t_1 <^u_r t_2)$.

Example 8.1.5 *Consider the preferences introduced in Example 7.1.1:*
$\langle\{RESIDENCE <^u_{\mathcal{D}} TIME <^u_{\mathcal{D}} SEX,$
$RESIDENCE <^u_{\mathcal{D}} TIME <^u_{\mathcal{D}} OCCUPATION,$
$RESIDENCE <^u_{\mathcal{D}} RACE <^u_{\mathcal{D}} SEX,$
$RESIDENCE <^u_{\mathcal{D}} RACE <^u_{\mathcal{D}} OCCUPATION\},$
$\{Male <^u_{SEX} Female\},$
$\{AvgIncome <^u_{Meas} AvgCostGas <^u_{Meas} AvgCostElec\}\rangle.$
And consider the two queries:
$q_1 = \{AllCities\} \times \{Female\} \times \{AllYears\} \times \{AllRaces\} \times \{AllOccs\}$
$q_2 = \{AllCities\} \times \{Male\} \times \{AllYears\} \times \{AllRaces\} \times \{AllOccs\}.$
It is $q_2 <^u_R q_1$ *since* $\langle AllCities, Male, AllYears, AllRaces, AllOccs\rangle <^u_r \langle AllCities, Female, AllYears, AllRaces, AllOccs\rangle$.

It is shown in [81] that $<^u_R$ is a total order if $<^u_r$ is a total order. A total order for $<^u_R$ is needed to find a unique subquery to the user query, since a partial order would not allow to distinguish between incomparable preferred subqueries. Note that a total order consistent with $<^u_r$ can be obtained from a partial order by simply composing the partial order $<^u_r$ with a given total order (e.g., the alphabetic or lexicographic order) through prioritization.

The problem to find the preferred subquery of a user query q can now be formally defined. Let q be a query in Q_{MDX}. Given user preferences $\mathcal{U} = \langle <^u_{\mathcal{D}}, \{<^u_{D_1}, \ldots, <^u_{D_N}, <^u_{Meas}\}\rangle$ defining a total order $<^u_R$ over the set of queries, and an anti-monotone constraint $v \in \mathcal{V}$, the problem is to find the most interesting subquery q^* of q such that q^* satisfies the constraint v, i.e. $v(q^*) = true$. Formally, the problem is to find the query q^* defined by:

$$q^* = max_{<^u_R}\{q' \in Q_{MDX} \mid q' \sqsubseteq q \wedge v(q') = true\}.$$

The algorithm for computing the preferred query, Algorithm 8, relies on the fact that a query is modeled as a set of references in the cube. It uses Function $NextRef$ defined below, to find the next preferred reference regarding the order over references induced by the user preferences.

Definition 8.1.2 - Function $NextRef$**.** *Let R be a set of references and $<_r$ a total order over R, $NextRef$ is defined, for all reference $x \in R$ as: $x' = NextRef_{<_r}(x, R)$ if $x' <_r x$, $(\nexists x'' \in R)(x' <_r x'' <_r x)$ and $x \neq min_{<_r}(R)$. If $x = min_{<_r}(R)$, $NextRef_{<_r}(x, R) = null$.*

Algorithm 8 Compute maximal subset

Input: R: A set of reference; $<_r$: A total order over R; v: A visualization constraint
Output: A maximal set of reference satisfying v
Variables:
1: $X = \emptyset$
2: $x = max_{<_r}(R)$
3: **while** $(x \neq null)$ **do**
4: **if** $v(X \cup \{x\}) = true$ **then**
5: $X = X \cup \{x\}$
6: $x = NextRef_{<_r}(x, R)$
return X

Soundness of this algorithm is shown in [81].

Example 8.1.6 *Consider again the preferences introduced in Example 7.1.1, with the query $q = \{AllCities\} \times \{Female, Male\} \times \{2011, 2010\} \times \{AllRaces\} \times \{AllOccs\}$. Suppose the constraint v is defined by $v(q) = true \equiv |q| \leq 2$. Then Algorithm 8 constructs $q^* = \{AllCities\} \times \{Female\} \times \{2011, 2010\} \times \{AllRaces\} \times \{AllOccs\}$.*

Deriving preference constructors

Once the subquery q^*, i.e., the preferred set of references, is obtained, it is used to derive the preference constructors to be added to the current query, as follows. The principle is that this subquery q^* corresponds to the un-dominated references of the user query q. The preference constructs are derived as follows, and composed with Pareto:

- If all the members of a level L of a hierarchy h appear in q^*, then derive a CONTAIN(h,L) construct.

- For the set V of members for level a appearing in q^*, derive a POS(a,V) construct.

Note that if the order induced by the preferences was over cell schemata instead of cell references, then the CONTAIN(measures,m) construct could be derived. Besides, if the query model used was that of fully evaluated queries, then the BETWEEN(m, v_{low}, v_{min}) construct could be derived as well.

Example 8.1.7 *Consider Example 8.1.6. Query q can be expressed in MDX by:*

```
SELECT {SEX.Sex.[Male],SEX.Sex.[Female]} ON COLUMNS,
       {TIME.Year.[2011],TIME.Year.[2010]} ON ROWS
FROM CENSUS
```

The MYMDX *query corresponding to* q^* *can be expressed by:*

```
SELECT {SEX.Sex.[Male],SEX.Sex.[Female]} ON COLUMNS,
       {TIME.Year.[2011],TIME.Year.[2010]} ON ROWS
FROM CENSUS  PREFERRING Sex POS 'Female'
```

8.2 Query expansion

This section focuses on the second type of personalization approaches, where query expansion is used to enrich the user query with preferences, without relying on dedicated operators. The principle is to reduce the answer to a user query by computing the preferred subquery in the classical sense of query inclusion. Such approaches are prescriptive by nature, and usually require low formulation effort. They are presented into three categories, according to whether the current query is unevaluated, partially evaluated, or fully evaluated.

8.2.1 Using unevaluated queries

The principle introduced in [64] can be directly applied: the simple preferences extracted over members (see Section 7.1 in Chapter 7) can be used to generate a user profile consisting of weights on selection conditions over members. This profile could be exploited using the algorithm described in [64].

Alternatively, if the profile consists in the navigational habits, the principle presented in Section 8.1.2 can be used. Instead of deriving preference constructors from the relevant habits, these habits are used to enforce hard constraints, or to discard group by sets. In the approach presented Section 8.1.2, only the fragment translation step has to be adapted. It should now consist in the following, according to the query fragment used for personalization:

- if the fragment is a selection predicate, propagate this predicate to the user query,
- if the fragment is a level in a hierarchy, discard the other levels of this hierarchy that the query features,

- if the fragment is a measure, discard the other measures that the query features.

Note that, in that case, preferences correspond to hard constraint combined using intersection, whereas in the previous section, they correspond to soft constraints combined using Pareto.

Example 8.2.1 *Consider the* MyMDX *query of 8.1.3. Using a query expansion approach, with the same preferences as the ones used in the annotations of this query, the resulting MDX query would be:*

```
SELECT Filter(OCCUPATION.members,
        AvgIncome BETWEEN 500 AND 1000) ON COLUMNS,
        Crossjoin((RACE.Mrn,
                Descendants(RESIDENCE.AllCities,RESIDENCE.Region))) ON ROWS
FROM CENSUS WHERE TIME.Year.[2009]
```

8.2.2 Using partially evaluated queries

If queries are modeled as sets of references, the subquery of the current query is directly the one computed using the principle given in Section 8.1.3. The MDX expression can straightforwardly be derived from the set of references by cross-joining the sets of members in each dimension that appear in the subquery.

Example 8.2.2 *Consider the* MyMDX *query of 8.1.6. Using a query expansion approach with the same preferences as the ones used in Example 8.1.6, the resulting MDX expression would be:*

```
SELECT {SEX.Sex.[Female]} ON COLUMNS,
        {TIME.Year.[2011],TIME.Year.[2010]} ON ROWS
FROM CENSUS
```

8.2.3 Using fully evaluated queries

If the current query is fully evaluated, i.e., corresponds to a set of facts, investigations extracted from the user's former sessions (see Section 7.3 of Chapter 7) can be used to expand the current query by restricting the set of cells to those showing information related to the discoveries detected in the log. Roughly speaking, the expanded query would somehow be the result of the discovery driven operators used on its cells.

More precisely, the difference pairs of the current query answer q are extracted and compared to the most general difference pairs of each investigations extracted from the log. For each investigation $i = \langle\langle c, c'\rangle, M, D, E\rangle$ such that a difference pair of the current query is a drill-down pair of $\langle c, c'\rangle$, the current query q is expanded as follows: $q' = \bigcup_{m \in M}(q \cap m) \cup \bigcup_{d \in D}(q \cap d) \cup \bigcup_{e \in E}(q \cap e)$. Recall that in this case, a query is modeled as a set of facts and consequently it may not be expressed as the result of a cross product.

Example 8.2.3 *Consider Example 7.3.1 and Example 7.3.5 where the investigations extracted from the log were* $i_1 = \langle p_1, \{q_5\}, \{q_2, q_4\}, \{q_1\}\rangle$ *and* $i_2 = \langle p_2, \{q_5\}, \{q_2, q_4\}, \emptyset\rangle$ *with*

$p_1 = \langle\langle AllYears, Female, Pacific, 100\rangle, \langle AllYears, Female, Atlantic, 200\rangle\rangle$ *and*

$p_2 = \langle\langle AllYears, Female, Pacific, 100\rangle, \langle AllYears, Male, Pacific, 200\rangle\rangle$ *and*

q_1		L.A.	N.Y.
2007	Female	200	100
	Male	200	250

q_2		L.A.	N.Y.
2008	Female	100	200
	Male	200	250

q_4		Pacific	Atlantic
2008	Female	100	200
	Male	200	250

q_5	Pacific	Atlantic
Female	100	200
Male	200	200

Consider also the current query:

$q_{current}$		Pacific	Atlantic	California	New York	L.A.	N.Y.
2008	Female	100	200	100	200	100	200
	Male	200	250	150	250	200	250
2007	Female	150	150	100	200	200	100
	Male	200	250	200	250	200	250
AllYears	Female	100	200	150	150	150	150
	Male	200	200	200	250	200	250

Expanding this current query using investigation $i_2 = \langle p_2, \{q_5\}, \{q_2, q_4\}, \emptyset\rangle$ *would result in the query:*

$q_{current}$		Pacific	Atlantic	L.A.	N.Y.
2008	Female	100	200	100	200
	Male	200	250	200	250
AllYears	Female	100	200		
	Male	200	200		

8.3 Conclusion

This chapter introduced various approaches for personalizing OLAP queries leveraging the knowledge extracted from query logs. The approaches are either prescriptive, requiring a low formulation effort, or not prescriptive and expressive, while demanding a higher formulation effort.

Importantly, these approaches are consistent with the requirements stated in the conclusion of Chapter 2. In particular, (i) they rely on user navigational habits extracted from past analytical sessions, (ii) the profile elements used respect the multidimensional model, and (iii) they can take advantage of the user's past discoveries in the data.

The approaches presented in the present chapter assume the log of a single user. The next chapter show how to complement these approaches with query recommendations, using a multi-user log.

Chapter 9

Collaborative recommendations with a multi-user log

This chapter presents various query recommendation techniques using information extracted from an OLAP query log. Though all the approaches presented here are inherently history-based, we use the categorization introduced in Chapter 2 Section 2.4 to distinguish the approaches that are hybrid in the sense that they are both history-based and current state (Section 9.1) from those that are purely history-based (Section 9.2). Most of the materials used in this chapter comes from the thesis of Elsa Negre [83], which have been published in [36], [37], [38] and [39].

9.1 An history-based and current state approach

In this section, we describe a query recommendation approach that uses both the information extracted from the log and the current state of the database, more precisely, the content of the fully evaluated query. In this approach, the profile is a set of investigations extracted from a multi-user log.

9.1.1 Leveraging past investigations

Recommendations are computed on the basis of the differences discovered in the log. The key idea is to detect the difference that the current query is investigating and to recommend the queries in the log that investigated the same difference. Recall from Chapter 7 that an investigation records the set of *mgdq* (the most general difference queries, i.e., the queries containing a difference pair at the coarsest granularity level in the log) and, at a lower level of detail, the

queries that confirm the difference (the *mgdq*'s drill-down differences), and the queries that contradicts the difference (their exceptions). Recommendations are computed online each time a current query is added to the current session by the current user. The current query is analyzed to detect to which investigations it corresponds (this query may be itself a *mgdq*, a drill-down difference, or an exception of what is detected in the log). Then a navigation plan (a set of queries arranged in a graph) is proposed for the current user to see drill-down differences or exceptions to the *mgdq*, by using the queries of the investigations.

Example 9.1.1 *We recall from Example 7.3.1 and Example 7.3.5 that the investigations extracted from the log were* $i_1 = \langle p_1, \{q_5\}, \{q_2, q_4\}, \{q_1\}\rangle$ *and* $i_2 = \langle p_2, \{q_5\}, \{q_2, q_4\}, \emptyset\rangle$ *with:*

$p_1 = \langle\langle AllYears, Female, Pacific, 100\rangle, \langle AllYears, Female, Atlantic, 200\rangle\rangle$

$p_2 = \langle\langle AllYears, Female, Pacific, 100\rangle, \langle AllYears, Male, Pacific, 200\rangle\rangle$

and:

q_1		*L.A.*	*N.Y.*
2007	*Female*	*200*	*100*
	Male	*200*	*250*

q_2		*L.A.*	*N.Y.*
2008	*Female*	*100*	*200*
	Male	*200*	*250*

q_4		*Pacific*	*Atlantic*
2008	*Female*	*100*	*200*
	Male	*200*	*250*

q_5	*Pacific*	*Atlantic*
Female	*100*	*200*
Male	*200*	*200*

These two investigations will be used to compute recommendations to current queries.

9.1.2 Computing and presenting recommendations

Given a current query q and a set of investigations I, the recommender system is implemented by Algorithm 9, that uses the operators defined in Section 7.3. It first identifies in I the set of *mgdp* to which q can be related. For such an *mgdp* m, q can be either a drill-down difference query w.r.t. m, a roll-up difference query w.r.t. m, or an exception difference query w.r.t. m. In each case a specific function is used to construct the recommendation from the investigation whose difference pair is m.

Function *recommendDrilldown* is given by Algorithm 10. The idea is to recommend each session which mgdq is a rollup difference query of the current query, with the queries of the session arranged in a given order (first the mgdq, then the drilldown differences of the current query, etc.). The recommendation is a navigation plan, i.e., a graph of queries rooted in the current query q (see Figure 9.1). The other functions used in Algorithm 9 for recommending sessions follow the same general principle. For instance, if the current query q is detected as an exception of the mgdq of an investigation, then it makes sense to present first the exceptions of the mgdq that are the rollup difference queries of q, and then the exceptions of the mgdq that are drilldown difference queries of q, since it will probably help the user to understand to which extends q contributes to being an exception of the mgdq.

Algorithm 9 Recommendations for a current query

Input: q: The current query, I: A set of investigations, fdp: A Boolean function
Output: A graph G of recommended queries
Variables: G: A graph , C: A set of queries

$G = \langle \emptyset, \emptyset \rangle$
$M = mgdq(I)$
for each difference pair $\langle c, c \rangle'$ of difference(q,fdp) **do**
  $C = rollupDifference(c, c', M)$ ▷ first check if q is a drill-down difference
  if $C \neq \emptyset$ **then**
    $G = G \cup recommendDrilldown(c, c', q, C)$
  $C = drilldownDifference(c, c', M)$ ▷ then check if q is a rollup difference
  if $C \neq \emptyset$ **then**
    $G \leftarrow G \cup recommendRollup(c, c', q, C)$
  for each difference pair $\langle x, x' \rangle$ of $m \in M$ **do** ▷ finally check if q is an exception
    $C = exceptions(x, x', \{q\})$
    if $C \neq \emptyset$ **then**
      $G = G \cup recommendException(c, c', q, C)$
return G

Function 10 recommendDrilldown

Input: $\langle c, c' \rangle$: A difference pair, q: A current query, C: A set of difference pairs
Output: A graph G of recommended queries
Variables: E: A set of edges, V: A set of vertices

$E = \emptyset$
$V = \emptyset$
for each pair $m \in C$ **do**
  let $i = \langle m, M, D, X \rangle$ ▷ get the investigation having m as mgdp
  $V = V \cup M \cup D \cup X$ ▷ create vertices and then create edges
  let $S1 = drilldownDifference(c, c', D)$ ▷ the drill-down differences of q
  let $S2 = exceptions(c, c', M)$ ▷ the exceptions to q in M
  let $S3 = D \setminus S1$ ▷ the drill-down diff. to m which are not drill-down diff. of q
  let $S4 = E \setminus S2$ ▷ the exceptions to m which are not exceptions of q
  $E = (q \times M) \cup (M \times M)$ ▷ Link q to the most general queries M
  $E = E \cup (M \times S1) \cup (S1 \times S1)$ ▷ Link the most general queries to the drill-down diff. of q
  $E = E \cup (S1 \times S2) \cup (S2 \times S2)$ ▷ Link the drill-down diff. of q to the exceptions to q
  $E = E \cup (S2 \times S3) \cup (S3 \times S3)$ ▷ Link the exceptions of q to the drill-down diff. to m
  $E = E \cup (S3 \times S4) \cup (S4 \times S4)$ ▷ Link the drill-down diff. of m to the exceptions to m
return $G = \langle V, E \rangle$

Example 9.1.2 *Consider the following current query:*

$q_{current}$		*California*	*New York*
2008	*Female*	*100*	*200*
	Male	*150*	*250*

We call Algorithm 9 with parameters: $q_{current}$ *(the current query),* $\{i_1, i_2\}$ *(the investigations extracted from the log recalled in the previous example) and the function* fdp *that outputs true for two cells* c, c' *if their measures differ by at least a factor of 2. It is detected that the pair* $p_1 = \langle\langle 2008, Female, California, 100\rangle, \langle 2008, Female, NewYork, 200\rangle\rangle$ *of* q_8 *is a drilldown pair of* $p_1 = \langle\langle AllYears, Female, Pacific, 100\rangle, \langle AllYears, Female, Atlantic, 200\rangle\rangle$ *and thus* $q_{current}$ *is a drilldown difference. Function 10 is used to construct the following graph:* $\{q_{current}, q_1, q_2, q_4, q_5\}, \{(q_{current}, q_5), (q_5, q_2), (q_5, q_4), (q_2, q_1), (q_4, q_1)\}$. *Using this graph, the user can navigate first to* q_5*, that will show that the difference observed in the average income of Females between California and New York is somehow not surprising in the sense that it is also valid for regions Pacific and Atlantic, whatever the year. Then, the user can navigate to* q_2 *and*

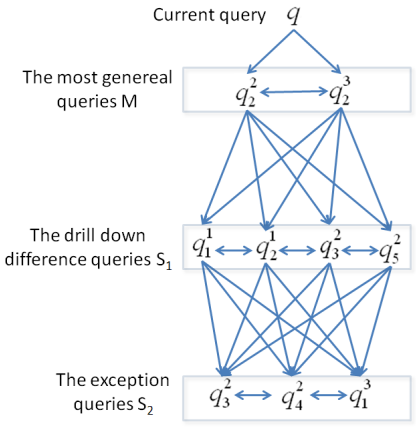

Figure 9.1: A navigation plan

q_4 to see other instances of this difference. Finally, the user can navigate to the only exception to this observation by looking at query q_1.

9.2 Pure history-based approaches

We now describe query recommendation approaches that use only the history of the database, that is the unevaluated queries of the log.

9.2.1 Leveraging similar sessions

In this section, we present a general framework for computing history-based recommendations, where a query is recommended by using both the server log, i.e., the set of sessions already posed to navigate the cube, and the sequence of queries of the current session. Note that this framework can be used whatever the query model (unevaluated, partially evaluated, fully evaluated), by adapting the similarity used for queries (and thus sessions). However, since Section 9.1 already presented an approach being both history-based and current state (based on fully evaluated queries), in this section we focus on pure history-based approaches, where the query model is that of unevaluated query.

The framework consists in three steps, as illustrated by Algorithm 11: (i) Matching, i.e., searching the log for sessions related to the current session, (ii) Predicting, i.e., generating candidate recommendations by predicting what could be the next query, and (iii) Ranking, i.e., ordering the candidate recommendations to present the queries to the user in interest order. This framework is flexible in the sense that it does not lay down a specific method for matching, predicting or ranking. Instead of that, each of these steps is parametrized with functions and relations. By changing these parameters, the way recommendations are computed changes. We now present these steps into more details.

Algorithm 11 Recommending leveraging similar sessions

Input: S_c: The current session, L: The query log, θ: A relation over sessions, $Pred$: A prediction function, $Ranking$: A query ranking function, $\prec$: An order over queries, $Default$: A function that returns a default recommendation.
Output: An ordered set of recommended queries
Variables: $CandSessions$: A set of sessions, $CandQ$: A set of queries

```
1: CandSessions ← σ_{θ,S_c}(L)                          ▷ Matching
2: CandQ ← Pred(S_c, CandSessions)                       ▷ Predicting
3: if CandQ ≠ ∅ then
4:     return Ranking(CandQ, ≺)                          ▷ Ranking
5: return Default(L, ≺)
```

Matching: Looking for similar ways of querying

The first step of the approach consists of using the selection operator on sessions defined in Section 5.2 of Chapter 5, to return a set of sessions related to the current session. The underlying idea is to find among the sessions of the log, those that are related to the current session and to recommend one of these queries to the user. As indicated in Section 5.2, the θ relation can use one of the similarities presented in Chapter 6 to compare the current session S_c with each session of the log, and output the closest to S_c or the top-k most similar. As session similarities can rely on a query similarity, any of the query similarities introduced in Chapter 6 can be used. Obviously the choice of the query similarity and session similarity to used is related to the query model used, but may also be related to the importance attached to the frequency of queries in the log or the importance of order of the queries in the session. For instance, if the order of queries is relevant but the frequency is not, then the Levenshtein distance combined with the Hausdorff distance can be used for the θ relation, as in [37].

Example 9.2.1 *Consider a log consisting of two sessions,* $s' = \langle q_4, q_5, q_6, q_7, q_8\rangle$, *and* $s'' = \langle q_9, q_{10}\rangle$, *and let* $s = \langle q_1, q_2, q_3\rangle$ *be the current session. Table 9.2.1 represents each single query in terms of the unevaluated query model; the involved group-by sets are:*

$$G_1 = \langle \mathsf{State}, \mathsf{Race}, \mathsf{Year}, \mathsf{AllSex}, \mathsf{Occ}\rangle$$
$$G_2 = \langle \mathsf{State}, \mathsf{RaceGroup}, \mathsf{Year}, \mathsf{AllSex}, \mathsf{Occ}\rangle$$
$$G_3 = \langle \mathsf{Region}, \mathsf{AllRaces}, \mathsf{Year}, \mathsf{Sex}, \mathsf{Occ}\rangle$$

while the selection predicates are:

$$c_1 = \{TRUE_{\mathsf{RESIDENCE}}, \ldots, (\mathsf{Year} = 2005), \ldots, TRUE_{\mathsf{SEX}}\}$$
$$c_2 = \{TRUE_{\mathsf{RESIDENCE}}, (\mathsf{RaceGroup} = Chinese), \ldots, \ldots, TRUE_{\mathsf{SEX}}\}$$
$$c_3 = \{TRUE_{\mathsf{RESIDENCE}}, (\mathsf{RaceGroup} = Chinese), (\mathsf{Year} = 2005), \ldots, TRUE_{\mathsf{SEX}}\}$$

Suppose algorithm 11 is called with this log, s *as the current session, and* $\theta(s, s') \equiv sim(s, s') \geq 0.5$ *where sim is the extension of Tf-Idf introduced in*

		Queries									
		q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_{10}
Group-by set		g_1	g_2	g_2	g_2	g_2	g_3	g_3	g_2	g_1	g_1
Measures	AvgCostWatr	✓	✓	✓	✓	✓	✓	✓	✓		
	AvgCostElect	✓	✓	✓		✓	✓		✓		
	AvgCostGas			✓						✓	✓
	AvgIncome							✓	✓		✓
Selection predicates		c_1	c_1	c_1	c_2	c_3	c_1	c_1	c_1	c_1	c_1

Table 9.1: Queries for Example 9.2.1

Chapter 6. $CandSessions = \{s'\}$ since $sim(s, s') = 0.673$ and $sim(s, s'') = 0$ (because queries similar to q_3 and q_8 can be found in each session of the log).

Predicting: choosing the queries to recommend

In the second step of the approach, Function *Pred* is used to obtain, from the set of sessions matching the current session, queries to recommend. In [37], we propose to return the last query of each candidate session, since this query, being the point where the user stopped the analysis, is likely to contain relevant information for the session. Note that this step can be expressed using the language for manipulating logs, as shown in Section 5.3.3.

We briefly list some other possibilities for this function:

- Summarize the queries of each candidate session by one single query, respectively, using the approach described Section 5.3.1.
- Return the query of each candidate session that is the closest to the last query of the current session.
- Return the query of each candidate session that is the best query in the sense of a preference relation over queries.

These various possibilities for the *Pred* function may lead to very diverse recommended queries. As in other contexts where recommendation is used, like Information Retrieval, diversity in recommendation is probably also very relevant for the OLAP user, and it may be desirable for a recommender system to use more than one *Pred* function.

As previously noted, the set of candidate recommendations can be empty. In that case, it might be useful to still be able to provide the user with a default recommendation. Thus,the function $Default(\mathcal{L})$ allows to obtain such a recommendation directly from the log. Various default recommendations can be proposed to the user. Considering that our set of sessions of the log can be seen as a graph where each node is a query and where the edges are the sequence of queries, borrowing an idea from [63] we propose as a default recommendation the nodes being the authorities (resp., the hubs), i.e., the nodes that have the highest number of successors (resp., predecessors).

Ranking the recommendations

During the previous step, a set of candidate queries to recommend is computed. The aim of this final step is to rank the queries, given a satisfaction criteria

or order relation $\prec$, that can be expressed by the user. The ranking function $Ranking(CandQ, \prec)$ returns the ordered set of recommended queries. Again there are many ways of ranking the candidates, from very basic to sophisticated ones. Recommendations can be ranked according to how close they are from the last query of the current session, by their number of references not already seen by the user in the current session, or by using preferences expressed over queries. In particular, for partially evaluated queries, the order $<_R^u$ stemming from the user profile (see Chapter 8) can be used for $\prec$. Using the language for log manipulation introduced in Chapter 5, $Ranking(CandQ, \prec)$ can be expressed by $agg_{\theta,agg}(L)$ where:

- $CandQ$ is turned into a log L by having each query converted into a session of length 1.
- $\theta(s, s')$ holds for all $s, s' \in L$,
- $agg(S) = \langle q_1, \ldots, q_n \rangle$ where for all $i, j \in \{1, n\}$, it is $i < j$ if $q_i \prec q_j$ holds.

Example 9.2.2 *Continuing Example 9.2.1, suppose Algorithm 11 is called with, for both Pred and Ranking, a function choosing the query closest to q_3 (the last query of the current session), in the sense of the similarity between unevaluated queries defined in Chapter 6. It can easily be seen that $candQ = \{q_8\}$, which is the query closest to q_3. This query is returned as the recommendation.*

9.2.2 Leveraging navigational habits

Navigational habits can be extracted from a multi-user query log (see Chapter 7). This knowledge can be exploited to derive recommendations, disregarding former sessions, in a spirit similar to the one used in Chapter 8, Section 8.1.2. In this case, the set of rules is searched for pertinent rules, i.e., whose antecedent matches the user query. The main difference is that the rules need not be effective, since the goal is not to reduce the answer set. Then pertinent rules' consequents are used to derive the recommendation.

Rule selection

Recall that the semantics of the rule describing a navigational habit is: if the antecedent appears in the query then the consequent tends to appear together with it. Hence the query recommended should be constructed starting from the current query, and should not be a subquery of it.

The selection of the rules follows the principle of Algorithm 5, with the differences that:

- The consequents selected need not be part of the current query.
- Parameter α is not needed any more, since what is needed to construct the recommended query is, at least, one fragment for the query group-by set and one fragment for the measures.

Algorithm 12 is an adaptation of Algorithm 5 in that sense, where $score(r)$ for a rule r being, e.g., its confidence.

Algorithm 12 Select fragments for recommending

Input: R: A set of rules; q: A query represented as a qf-set
Output: F: A qf-set that will be used as a recommendation
Variables: F: A qf-set

1: $R = R \setminus \{r \in R | r.ant \not\sqsubseteq q\}$ ▷ Drop non-pertinent rules
2: $R = R \setminus \{r \in R | r.cons \in A \cup M, r.cons \in q\}$ ▷ Drop effective rules
3: $F = \emptyset$
4: $end = false$
5: **while** $end = false$ and $R \neq \emptyset$ **do** ▷ Iteratively construct the answer...
6: let $r = argmax_R\ score(r)$ ▷ ...starting with the rule having highest score
7: $R = R \setminus \{r\}$
8: $F = F \cup \{r.cons, r.ant\}$
9: **if** $\exists \varphi \in F | \exists h \in H, \varphi \in Lev(h)$ and $\exists \varphi' \in F | \varphi' \in M$ **then**
10: $end = true$
11: **return** F

Deriving recommendation

A basic idea developed here is to construct the recommended query using only the following fragments: 1) those of the user query that appear as antecedents of the selected rules, and 2) those appearing as consequents of the selected rules. If a group by set cannot be formed because some hierarchies do not appear in the qf-set, then the qf-set is completed with the coarsest levels of these hierarchies.

Example 9.2.3 *Consider the qf-set describing the current query* $q = \{$Region, AllCities, Mrn, AllRaces, Occ, Year, AllSexes, AvgIncome, (Year $= 2009)\}$. *Let the set* R *of rules extracted from the log be as follows:*

r_1: (Region $\in \{$*'Pacific','Atlantic'*$\}$) $\rightarrow$ Year *(0.8)*
r_2: Year $\rightarrow$ Region *(0.80)*
r_3: Year $\rightarrow$ AllCities *(0.60)*
r_4: AvgIncome $\rightarrow$ Cities *(0.60)*
r_5: Year $\rightarrow$ Sex *(0.90)*
r_6: (Year $= 2009$) $\rightarrow$ Region *(0.70)*
r_7: Year $\rightarrow$ (Year $\in 2009$) *(0.50)*
r_8: Year $\rightarrow$ (AvgIncome $\in [500, 1000]$) *(0.55)*
r_9: AvgIncome $\rightarrow$ Mrn *(0.45)*
r_{10}: Occ $\rightarrow$ Region *(0.70)*
r_{11}: Occ $\rightarrow$ Year *(0.10)*
r_{12}: AvgIncome $\rightarrow$ Year *(0.70)*

Algorithm 12 outputs the qf-set {AvgIncome, Cities, Year, Sex, (AvgIncome $\in [500, 1000]$)}, *using only rules* r_4, r_5, r_8 *that are pertinent and non effective. This qf-set is completed with the coarsest levels of the missing hierarchies, resulting in* {AvgIncome, Cities, Year, Sex, AllOccs, AllRaces, (AvgIncome $\in [500, 1000]$)}.

9.3 Conclusion

This chapter introduced various approaches for recommending OLAP queries based on the knowledge extracted from multi-user query logs. The approaches

either rely on the log and on the database instance, and in that case the recommended queries indicate potential discoveries made by other users analyzing a related set of data, or they exploit past unevaluated queries only, by looking for similar user intensions or navigational habits.

As for the personalization approaches presented in the previous chapter, the recommendation approaches are aligned with the requirements stated in Chapter 2, being tailored for the multidimensional model and the ways users analyze a cube.

The following chapter concludes the dissertation, with a brief report on the experiments conducted to assess the approaches introduced in the chapter and the previous one, and draws research perspectives.

Part V

Conclusion

Chapter 10

Towards analytical sessions of better quality

This chapter concludes the dissertation by summarizing the contributions (Section 10.1) and discussing future research directions (Section 10.2). It relies partly on submitted material [14] following a Dagstuhl Seminar on Data Warehousing [98].

10.1 Summary

10.1.1 The contributions

This dissertation is a contribution to developing user-centric OLAP, focusing on the use of former queries logged by an OLAP server to enhance subsequent analyses. It showed how logs of OLAP queries can be modeled, constructed (Part II), manipulated, compared (Part III), and finally leveraged for personalization and recommendation (Part IV).

Logs are modeled as sets of sessions, sessions being modeled as sequences of OLAP queries. Three main approaches are presented for modeling queries: as unevaluated collections of fragments (e.g., group by sets, sets of selection predicates, sets of measures), as sets of references obtained by partially evaluating the query over dimensions, or as query answers.

Such logs can be constructed even from sets of SQL query expressions, by translating these expressions into a multidimensional algebra, and bridging the translations to detect analytical sessions.

Logs can be searched, filtered, compared, combined, modified and summarized with a language inspired by the relational algebra and parametrized by binary relations over sessions. In particular, these relations can be specialization relations or based on similarity measures over queries and sessions.

Logs can be mined for various hidden knowledge, that, depending on the query model used, accurately represents the user behavior extracted. This

		Model of query		
		Unevaluated	Partially evaluated	Fully evaluated
Personalization	Operator	Section 8.1.2, [9]	Section 8.1.3	
	Query expansion	Section 8.2.1	Section 8.2.2 [16], [81]	Section 8.2.3
Recommendation	Current & history			Section 9.1 [38], [39]
	History	Sections 9.2.1, 9.2.2	Section 9.2.1[37]	Section 9.2.1

Table 10.1: Summary of the contributions

knowledge can be used for query personalization, i.e., coping with a current query too few or too many results, or query recommendation, i.e., suggesting queries to pursue an analytical session. This is summarized in Table 10.1, that shows the contributions in terms of sections of this dissertations, publications, approaches (personalization or recommendation) and query models.

10.1.2 Assessing the contributions

We now briefly present the objective tests conducted to assess the contributions. The detailed results of the tests can be found in [81], [37], [9], [39] and [83]. Note that subjective tests with real logs and/or involving users, to tune the approaches, are part of the perspectives, as underlined below.

The implemented approaches have been developed using mostly open source technologies, especially Java and the Mondrian OLAP engine, often coupled with MySQL. The approach of [16, 81] was implemented as a web service, in a mobile context, where personalization was used to reduce the size of query answers to have them fit the screen of a mobile device.

The logs used where synthetic logs generated with various log generators we developed on purpose. As representative illustrations, we mention:

- The generators used in [39] and [9], that simulated discovery driven analyses. Each session of that log consists of queries obtained using the Icube operators[1] that either roll-up or drill-down to meaningful zone of a cube given the answer of the current query. A specific parameter rules the density of the log. It represents the number of dimensions that can be manipulated in a session to explore the cube. The higher this number, the higher the probability of exploring different parts of the cube, and hence the sparser the log.

- The generator used in [10], that produces pairs of sessions based on the templates depicted in Figure 10.1, that model intuitive notions of what similar sessions could be:

  - In template $\wedge$, the two sessions have similar starting queries then they diverge to radically different queries.
  - In template $\vee$, the two sessions have radically different starting queries then they converge to similar ending queries.

[1] http://www.it.iitb.ac.in/~sunita/icube/index.htm

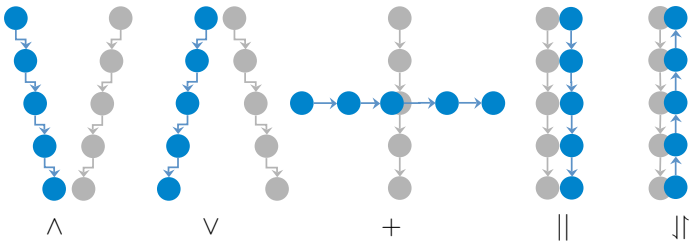

Figure 10.1: The templates used to generate sessions. Overlapping circles represent identical queries, near circles represent similar queries. For template ||, the queries are pairwise separated by one atomic OLAP operation

- In template +, the two sessions converge to the same query then they diverge.
- In template ||, the second session is constructed by "shifting" all queries in the first session by one OLAP operation.
- In template ⇃↾, the two sessions have the same queries in reverse order.

Regarding effectiveness, [81] and [9] report effectiveness in terms of reduction of the answer set using personalization techniques. [37] and [39] report effectiveness in terms of prediction of queries existing in a given part of a log when the technique is trained on other parts of this log. As expected, the log density, i.e., the fact that the log explores a small part of a cube, plays a significant role in the effectiveness of the approach. Indeed, the more dense the log, the more effective the approach.

Regarding efficiency, [37] and [39] showed that recommending an OLAP query can be computed efficiently for logs of reasonable sizes. [81] and [9] showed that personalization puts no significant overhead in the querying process, and that personalized queries are evaluated faster than non personalized queries.

10.1.3 Critical analysis of the contributions

This dissertation is a contribution to the huge task of developing user-centric OLAP, and as such is perfectible. We underline its main weaknesses, focusing on the foundations of query log management for user empowerment on the one hand, and on the confrontation of the proposed approaches on the other hand.

Regarding the first aspect, a better characterization of the complexity of the problems tackled in part IV (extracting knowledge from the log and using this knowledge for personalization and recommendation) is needed. These problems have been treated by essentially proposing algorithms and empirical tests, a more theoretical study remains to be conducted. Besides this characterization, the logical properties of log manipulation for user-centric OLAP should be investigated. This should be realized by using the language introduced in Chapter 5, that can be seen as a preliminary contribution to the foundations of OLAP query log management.

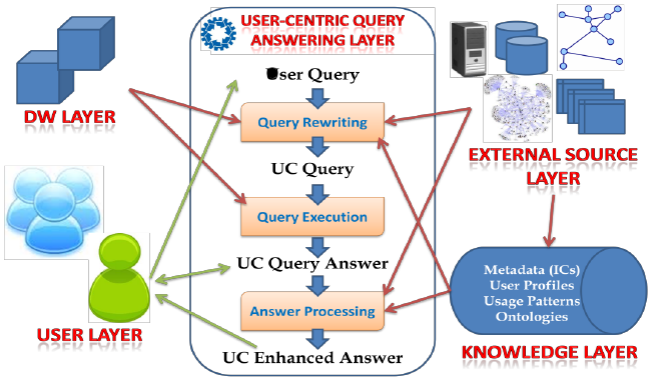

Figure 10.2: A user-centric query answering architecture

As to the second aspect, while the user-centric approaches proposed in this dissertations can obviously still be developed and improved, they have been devised mostly independently from each other. Investigating whether they can be combined or what approach better fits a given analytical context is still an open issue. For instance, it is open deciding if the most powerful approach for user empowerment is the one demanding a low formulation effort and being not prescriptive, proactive, highly expressive, both current state and history-based, while using knowledge external to the database. In that sense, a better modeling of the user's various analytical goals is still missing (the notion of investigation as defined in Chapter 7 can be seen as a first step towards such a model). Such a model would not only allow to assess the quality of analytical sessions, but also to track the evolution of the user's understanding while navigating a data warehouse.

This second aspect is developed in the following section.

10.2 Perspectives

The perspectives are presented starting from the short term ones, where is introduced an envisioned architecture for user-centric query answering (Section 10.2.1), to the longer term ones, focusing on quality of the analytical sessions (Section 10.2.2).

10.2.1 An envisioned architecture for user-centric query answering in data warehouses

The various techniques presented in the previous chapters could be integrated in a complete architecture for User-Centric Query Answering (UCQA) in data warehouses. In [14], such an architecture is envisioned (see Figure 10.2).

The UCQA Layer which is at the core of the architecture, is organized in a modular fashion, based on the following sub-components which handle specific

tasks of the whole advanced query answering phase.

Query Rewriting

This component takes as input the User Query (the query directly input via the user interface, that can be a set of keywords or expressed in natural language) and returns as output the User-Centric Query (UC Query, the query that can be evaluated directly by the data warehouse server).

In a data warehousing context, the end user is not expected to master a query language for expressing OLAP queries. Instead, the ubiquitous search engine interface should be favored [54]. The user-centric query expressed in the database language, SQL or MDX, should be deduced from the user keywords, profile, context, etc., by means of query rewriting paradigms that meaningfully combine multidimensional data stored in the target data warehouse, knowledge patterns from the External Source Layer, and, finally, knowledge components and constructs from the Knowledge Layer.

To this end, user-centric techniques like personalization and recommendation can be used to infer the UC Query [99, 103] from a keyword-based query. More precisely, in such a context, user empowerment should be achieved: 1) by using non prescriptive techniques, resulting in a flexible query expression, to help keyword and context disambiguation, 2) without demanding a high formulation effort, and 3) by using proactive suggestions. We briefly comment these options. With respect to the first aspect, a flexible query language, tailored for multidimensional queries, like the one proposed by [46], will enable the formulation of the UC Query without imposing hard constraints. Instead, user keywords and analytical context will be interpreted as soft constraints. In the second perspective, usage patterns capturing former traces (in particular, previous queries) should be leveraged for inferring the user's interests [9, 69, 103] and similarity with former sessions [39, 10] should help inferring the analytical context. Constraints imposed by the device should also be taken into account [16]. With respect to the third aspect, query suggestion can be used for generating the user-centric query as well as anticipating the forthcoming queries [39]. Coordinating these various approaches, in particular to rank the candidate UC Queries, will be facilitated with the use of a language that helps express both a data warehouse query and preferences, since a query in this language can be seen as an ordered set of classical queries [46].

Query Execution

This component is in charge of executing the UC query generated by the Query Rewriting component. To this end, the Query Execution component only accesses the target data warehouse where the multidimensional data of interest to the end-user are stored. The query rewriting process is obviously, transparent to the end-user and the query execution task produces as output the User-Centric Query Answer (UC Query Answer), the extensional answer that is retrieved by the data warehouse server in response to the UC query.

Answer Processing

The goal of the Answer Processing component consists in further enhancing the knowledge expressed by the UC query answer with some additional knowledge using cooperative query answering methods [79, 80]. As output, the User-Centric Enhanced Query Answer (UC Enhanced Query Answer, the answer augmented using cooperative techniques, to be presented under the form of dashboard) is finally provided to the end-user. To devise this User-Centric Enhanced Query Answer, data mining techniques on the extensional answer, internal (integrity constraints, usage patterns) and external knowledge should be used for complementing the answer with information outside the cube. These alternate answers can be ranked not only by leveraging the user history and preferences, but also by using criteria such as the answer's compactness or reliability. We note that, as no general model has been proposed for cooperative answers, finding the best form of the cooperative answer in a data warehouse context remains a challenging research problem. A first approach, tailored for OLAP sessions, is investigated in [72]. Another important research aspect related to cooperative query answering methods of the proposed user-centric data warehouse architecture, concerns the issue of meaningfully visualizing the so-retrieved results, in a concise and meaningful answer, even in an appealing graphical fashion.

10.2.2 Assessing the quality of analytical sessions

The previous section briefly described objective tests we performed to assess the effectiveness of the contributions. Subjective test involving users and user traces will have to be run. To conduct and simulate such tests, a perspective is the development of a platform for assessing the quality of an analytical session over a cube. Indeed, although data quality is a well studied area of data management, the quality of the querying process has not yet been investigated. Data quality is modeled according to quality dimensions (like e.g., accuracy.), each of them grouping quality factors (like e.g., semantic correction, or syntactic correction), each of them measured by various metrics [18]. A similar approach could be used to model the quality of analytical queries and the quality of analytical sessions. In addition, such a model would allow the development of a platform for assessing and validating user-centric approaches in the context of OLAP.

Notably, in domains like Information Retrieval, benchmarks are used [33] to validate approaches aiming at supporting browsing and exploratory search results. In the domain of data warehousing, the existing benchmarks are exclusively focused on performance (see for instance [104, 84, 30]), and effectiveness of the querying process in terms of how successful the analytical session is, is simply not addressed. So far, validating personalization or recommendation approaches is based on synthetic query logs [75] or the processing of existing logs with no guarantee of being realistic [24].

A platform for assessing user-centric approaches should allow to measure to which extent approaches are effective in that answers found are relevant, the effort spent to conduct the analysis is reduced, etc. Such a platform could benefit from previous effort to develop a benchmark to validate personalization

approaches in databases [87].

Finally, an even longer term perspective is extending OLAP to exploratory search. Exploratory search can be defined by an elaborated activity that goes beyond lookup and involves learning and investigating [73]. Modeling the overall decision making process under the form of exploratory search will allow to take advantage of how the user understanding of data evolves along her/his interaction with the data warehouse.

Bibliography

[1] Alberto Abelló, José Samos, and Fèlix Saltor. YAM^2: a multidimensional conceptual model extending UML. *Inf. Syst.*, 31(6):541–567, 2006.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] Gediminas Adomavicius and Alexander Tuzhilin. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.

[4] Gediminas Adomavicius, Alexander Tuzhilin, and Rong Zheng. REQUEST: A Query Language for Customizing Recommendations. *Information Systems Research*, 22(1):99–117, 2011.

[5] Rakesh Agrawal, Ashish Gupta, and Sunita Sarawagi. Modeling Multidimensional Databases. In *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997 Birmingham U.K*, pages 232–243, 1997.

[6] Rakesh Agrawal, Ralf Rantzau, and Evimaria Terzi. Context-sensitive ranking. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 383–394, 2006.

[7] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499, 1994.

[8] Javad Akbarnejad, Gloria Chatzopoulou, Magdalini Eirinaki, Suju Koshy, Sarika Mittal, Duc On, Neoklis Polyzotis, and Jothi Swarubini Vindhiya Varman. SQL QueRIE Recommendations. *PVLDB*, 3(2):1597–1600, 2010.

[9] Julien Aligon, Matteo Golfarelli, Patrick Marcel, Stefano Rizzi, and Elisa Turricchia. Mining Preferences from OLAP Query Logs for Proactive Personalization. In *Advances in Databases and Information Systems - 15th International Conference, ADBIS 2011, Vienna, Austria, September 20-23, 2011. Proceedings*, pages 84–97, 2011.

[10] Julien Aligon, Matteo Golfarelli, Patrick Marcel, Stefano Rizzi, and Elisa Turricchia. Similarity measures for OLAP sessions. *Submitted*, 2012.

[11] Julien Aligon, Haoyuan Li, Patrick Marcel, and Arnaud Soulet. Towards a logical framework for OLAP query log manipulation. In *PersDB 2012, 6th International Workshop on Personalized Access, Profile Management, and Context Awareness in Databases (invited paper)*, 2012.

[12] Julien Aligon, Patrick Marcel, and Elsa Negre. Résumés et interrogations de logs de requête OLAP. In *Extraction et gestion des connaissances (EGC'2011), Actes, 25 au 29 janvier 2011, Brest, France*, pages 239–250, 2011.

[13] Kamel Aouiche, Pierre-Emmanuel Jouve, and Jérôme Darmont. Clustering-Based Materialized View Selection in Data Warehouses. In *Advances in Databases and Information Systems, 10th East European Conference, ADBIS 2006, Thessaloniki, Greece, September 3-7, 2006, Proceedings*, pages 81–95, 2006.

[14] Marie-Aude Aufaure, Alfredo Cuzzocrea, Cécile Favre, Patrick Marcel, and Rokia Missaoui. Modeling and supporting user-centric query activities on data warehouses. *Submitted to IJDWM*, 2012.

[15] Eftychia Baikousi, Georgios Rogkakos, and Panos Vassiliadis. Similarity measures for multidimensional data. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 171–182, 2011.

[16] Ladjel Bellatreche, Arnaud Giacometti, Patrick Marcel, Hassina Mouloudi, and Dominique Laurent. A personalization framework for OLAP queries. In *DOLAP 2005, ACM 8th International Workshop on Data Warehousing and OLAP, Bremen, Germany, November 4-5, 2005, Proceedings*, pages 9–18, 2005.

[17] Henrike Berthold, Philipp Rösch, Stefan Zöller, Felix Wortmann, Alessio Carenini, Stuart Campbell, Pascal Bisson, and Frank Strohmaier. An architecture for ad-hoc and collaborative business intelligence. In *Proceedings of the 2010 EDBT/ICDT Workshops, Lausanne, Switzerland, March 22-26, 2010*, 2010.

[18] Laure Berti-Equille, Isabelle Comyn-Wattiau, Mireille Cosquer, Zoubida Kedad, Sylvaine Nugier, Verónika Peralta, Samira Si-Said Cherfi, and Virginie Thion-Goasdoué. Assessment and analysis of information quality: a multidimensional model and case studies. *IJIQ*, 2(4):300–323, 2011.

[19] Paolo Biondi, Matteo Golfarelli, and Stefano Rizzi. Preference-based datacube analysis with MYOLAP. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 1328–1331, 2011.

[20] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The Skyline Operator. In *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany*, pages 421–430, 2001.

[21] Ronen I. Brafman, Carmel Domshlak, Solomon Eyal Shimony, and Y. Silver. Preferences over sets. In *AAAI*, pages 1101–1106. AAAI Press, 2006.

[22] Benjamin Bustos and Tomás Skopal. Non-metric similarity search problems in very large collections. In Serge Abiteboul, Klemens Böhm, Christoph Koch, and Kian-Lee Tan, editors, *ICDE*, pages 1362–1365. IEEE Computer Society, 2011.

[23] Gloria Chatzopoulou, Magdalini Eirinaki, Suju Koshy, Sarika Mittal, Neoklis Polyzotis, and Jothi Swarubini Vindhiya Varman. The QueRIE system for Personalized Query Recommendations. *IEEE Data Eng. Bull.*, 34(2):55–60, 2011.

[24] Gloria Chatzopoulou, Magdalini Eirinaki, and Neoklis Polyzotis. Query Recommendations for Interactive Database Exploration. In *Scientific and Statistical Database Management, 21st International Conference, SSDBM 2009, New Orleans, LA, USA, June 2-4, 2009, Proceedings*, pages 3–18, 2009.

[25] Surajit Chaudhuri and Umeshwar Dayal. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26(1):65–74, 1997.

[26] Surajit Chaudhuri, Umeshwar Dayal, and Vivek R. Narasayya. An overview of business intelligence technology. *Commun. ACM*, 54(8):88–98, 2011.

[27] Pei-Yu Sharon Chen, Shin yi Wu, and Jungsun Yoon. The Impact of Online Recommendations and Consumer Feedback on Sales. In *Proceedings of the International Conference on Information Systems, ICIS 2004, December 12-15, 2004, Washington, DC, USA*, pages 711–724, 2004.

[28] Jan Chomicki. Preference formulas in relational queries. *ACM Trans. Database Syst.*, 28(4):427–466, 2003.

[29] William W. Cohen, Pradeep D. Ravikumar, and Stephen E. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03), August 9-10, 2003, Acapulco, Mexico*, pages 73–78, 2003.

[30] Jérôme Darmont, Fadila Bentayeb, and Omar Boussaid. Benchmarking data warehouses. *IJBIDM*, 2(1):79–104, 2007.

[31] Cláudio Rebelo de Sá, Carlos Soares, Alípio Mário Jorge, Paulo J. Azevedo, and Joaquim Pinto da Costa. Mining Association Rules for Label Ranking. In *Advances in Knowledge Discovery and Data Mining - 15th Pacific-Asia Conference, PAKDD 2011, Shenzhen, China, May 24-27, 2011, Proceedings, Part II*, pages 432–443, 2011.

[32] Françoise Fogelman-Soulié. Industrializing Data Mining, Challenges and Perspectives. In *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part I*, page 1, 2008.

[33] National Institute for Standards and Technology (NIST). Text retrieval conference (trec) home page. http://trec.nist.gov/, June 2012.

[34] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer D. Widom. *Database Systems: The Complete Book, Second edition.* Prentice Hall, 2008.

[35] Antara Ghosh, Jignashu Parikh, Vibhuti S. Sengar, and Jayant R. Haritsa. Plan Selection Based on Query Clustering. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, pages 179–190, 2002.

[36] Arnaud Giacometti, Patrick Marcel, and Elsa Negre. A framework for recommending OLAP queries. In *DOLAP 2008, ACM 11th International Workshop on Data Warehousing and OLAP, Napa Valley, California, USA, October 30, 2008, Proceedings*, pages 73–80, 2008.

[37] Arnaud Giacometti, Patrick Marcel, and Elsa Negre. Recommending Multidimensional Queries. In *Data Warehousing and Knowledge Discovery, 11th International Conference, DaWaK 2009, Linz, Austria, August 31 - September 2, 2009, Proceedings*, pages 453–466, 2009.

[38] Arnaud Giacometti, Patrick Marcel, Elsa Negre, and Arnaud Soulet. Query recommendations for OLAP discovery driven analysis. In *DOLAP 2009, ACM 12th International Workshop on Data Warehousing and OLAP, Hong Kong, China, November 6, 2009, Proceedings*, pages 81–88, 2009.

[39] Arnaud Giacometti, Patrick Marcel, Elsa Negre, and Arnaud Soulet. Query Recommendations for OLAP Discovery-Driven Analysis. *IJDWM*, 7(2):1–25, 2011.

[40] Arnaud Giacometti, Patrick Marcel, and Arnaud Soulet. A Relational View of Pattern Discovery. In *Database Systems for Advanced Applications - 16th International Conference, DASFAA 2011, Hong Kong, China, April 22-25, 2011, Proceedings, Part I*, pages 153–167, 2011.

[41] Sharad Goel, Andrei Z. Broder, Evgeniy Gabrilovich, and Bo Pang. Anatomy of the long tail: ordinary people with extraordinary tastes. In *Proceedings of the Third International Conference on Web Search and Web Data Mining, WSDM 2010, New York, NY, USA, February 4-6, 2010*, pages 201–210, 2010.

[42] Matteo Golfarelli. Handling Large Workloads by Profiling and Clustering. In *Data Warehousing and Knowledge Discovery, 5th International*

Conference, DaWaK 2003, Prague, Czech Republic, September 3-5,2003, Proceedings, pages 212–223, 2003.

[43] Matteo Golfarelli. Personalization of OLAP queries. In *6èmes journées francophones sur les Entrepôts de Données et l'Analyse en ligne (EDA 2010), Djerba, Tunisie*, volume B-6 of *RNTI*, page 1, Toulouse, Juin 2010. Cépaduès. Invited paper.

[44] Matteo Golfarelli and Stefano Rizzi. *Data Warehouse Design: Modern Principles and Methodologies*. McGraw-Hill, 2009.

[45] Matteo Golfarelli and Stefano Rizzi. Expressing OLAP Preferences. In *Scientific and Statistical Database Management, 21st International Conference, SSDBM 2009, New Orleans, LA, USA, June 2-4, 2009, Proceedings*, pages 83–91, 2009.

[46] Matteo Golfarelli, Stefano Rizzi, and Paolo Biondi. myOLAP: An Approach to Express and Evaluate OLAP Preferences. *IEEE Trans. Knowl. Data Eng.*, 23(7):1050–1064, 2011.

[47] Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total. In *Proceedings of the Twelfth International Conference on Data Engineering, February 26 - March 1, 1996, New Orleans, Louisiana*, pages 152–159, 1996.

[48] A. Gupta and I. Mumick. *Materialized views: techniques, implementations, and applications*. MIT Press, 1999.

[49] Ashish Gupta, Venky Harinarayan, and Dallan Quass. Aggregate-Query Processing in Data Warehousing Environments. In *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland*, pages 358–369, 1995.

[50] Marc Gyssens and Laks V. S. Lakshmanan. A Foundation for Multi-dimensional Databases. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 106–115, 1997.

[51] Mohand-Said Hacid, Patrick Marcel, and Christophe Rigotti. A Rule-Based Data Manipulation Language for OLAP Systems. In *Deductive and Object-Oriented Databases, 5th International Conference, DOOD'97, Montreux, Switzerland, December 8-12, 1997, Proceedings*, pages 417–418, 1997.

[52] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[53] Stefan Holland, Martin Ester, and Werner Kießling. Preference Mining: A Novel Approach on Mining User Preferences for Personalized Applications. In *Knowledge Discovery in Databases: PKDD 2003, 7th European Conference on Principles and Practice of Knowledge Discovery in Databases, Cavtat-Dubrovnik, Croatia, September 22-26, 2003, Proceedings*, pages 204–216, 2003.

[54] H. V. Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian, Yunyao Li, Arnab Nandi, and Cong Yu. Making database systems usable. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pages 13–24, 2007.

[55] Matthias Jarke, Maurizio Lenzerini, Yannis Vassiliou, and Panos Vassiliadis, editors. *Fundamentals of Data Warehouses (2nd Edition)*. Springer-Verlag, 2003.

[56] Abhijit Kadlag, Amol V. Wanjari, Juliana Freire, and Jayant R. Haritsa. Supporting Exploratory Queries in Databases. In *Database Systems for Advances Applications, 9th International Conference, DASFAA 2004, Jeju Island, Korea, March 17-19, 2004, Proceedings*, pages 594–605, 2004.

[57] Nodira Khoussainova, Magdalena Balazinska, Wolfgang Gatterbauer, YongChul Kwon, and Dan Suciu. A Case for A Collaborative Query Management System. In *CIDR 2009, Fourth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2009, Online Proceedings*, 2009.

[58] Nodira Khoussainova, YongChul Kwon, Magdalena Balazinska, and Dan Suciu. SnipSuggest: Context-Aware Autocompletion for SQL. *PVLDB*, 4(1):22–33, 2010.

[59] Nodira Khoussainova, YongChul Kwon, Wei-Ting Liao, Magdalena Balazinska, Wolfgang Gatterbauer, and Dan Suciu. Session-Based Browsing for More Effective Query Reuse. In *Scientific and Statistical Database Management - 23rd International Conference, SSDBM 2011, Portland, OR, USA, July 20-22, 2011. Proceedings*, pages 583–585, 2011.

[60] Werner Kießling. Foundations of Preferences in Database Systems. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, pages 311–322, 2002.

[61] Werner Kießling. Preference Queries with SV-Semantics. In *Advances in Data Management 2005, Proceedings of the Eleventh International Conference on Management of Data, January 6, 7, and 8, 2005, Goa, India*, pages 15–26, 2005.

[62] Ralph Kimball. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley, 1996.

[63] Jon M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *J. ACM*, 46(5):604–632, 1999.

[64] Georgia Koutrika and Yannis E. Ioannidis. Personalization of Queries in Database Systems. In *Proceedings of the 20th International Conference on Data Engineering, ICDE 2004, 30 March - 2 April 2004, Boston, MA, USA*, pages 597–608, 2004.

[65] Georgia Koutrika and Yannis E. Ioannidis. Personalized Queries under a Generalized Preference Model. In *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, pages 841–852, 2005.

[66] Laks V. S. Lakshmanan, Jian Pei, and Yan Zhao. QC-Trees: An Efficient Summary Structure for Semantic OLAP. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, pages 64–75, 2003.

[67] Hans-Joachim Lenz and Arie Shoshani. Summarizability in OLAP and Statistical Data Bases. In *Ninth International Conference on Scientific and Statistical Database Management, Proceedings, August 11-13, 1997, Olympia, Washington, USA*, pages 132–143, 1997.

[68] Wenmin Li, Jiawei Han, and Jian Pei. Cmar: Accurate and efficient classification based on multiple class-association rules. In Nick Cercone, Tsau Young Lin, and Xindong Wu, editors, *ICDM*, pages 369–376. IEEE Computer Society, 2001.

[69] Alexander Löser, Sebastian Arnold, and Tillmann Fiehn. The GoOLAP Fact Retrieval Framework. In Marie-Aude Aufaure, Esteban Zimányi, Wil Aalst, John Mylopoulos, Michael Rosemann, Michael J. Shaw, and Clemens Szyperski, editors, *Business Intelligence*, volume 96 of *Lecture Notes in Business Information Processing*. Springer Berlin Heidelberg, 2012.

[70] Andreas S. Maniatis, Panos Vassiliadis, Spiros Skiadopoulos, and Yannis Vassiliou. Cpm: A cube presentation model for olap. In Yahiko Kambayashi, Mukesh K. Mohania, and Wolfram Wöß, editors, *DaWaK*, volume 2737 of *Lecture Notes in Computer Science*, pages 4–13. Springer, 2003.

[71] Patrick Marcel. Olap query personalisation and recommendation: An introduction. In Marie-Aude Aufaure, Esteban Zimányi, Wil Aalst, John Mylopoulos, Michael Rosemann, Michael J. Shaw, and Clemens Szyperski, editors, *Business Intelligence*, volume 96 of *Lecture Notes in Business Information Processing*, pages 63–83. Springer Berlin Heidelberg, 2012.

[72] Patrick Marcel, Rokia Missaoui, and Stefano Rizzi. Towards intensional answers to OLAP queries for analytical sessions. In Il-Yeol Song and Matteo Golfarelli, editors, *DOLAP*. ACM, 2012.

[73] Gary Marchionini. Exploratory search: from finding to understanding. *Commun. ACM*, 49(4):41–46, 2006.

[74] Minnesota Population Center. Integrated public use microdata series. http://www.ipums.org, 2008.

[75] Chaitanya Mishra and Nick Koudas. Interactive query refinement. In *EDBT 2009, 12th International Conference on Extending Database Technology, Saint Petersburg, Russia, March 24-26, 2009, Proceedings*, pages 862–873, 2009.

[76] Tom M. Mitchell. Generalization as Search. *Artif. Intell.*, 18(2):203–226, 1982.

[77] Alvaro E. Monge and Charles Elkan. An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate Database Records. In *DMKD*, pages 0–, 1997.

[78] Erwan Moreau, François Yvon, and Olivier Cappé. Robust Similarity Measures for Named Entities Matching. In *COLING 2008, 22nd International Conference on Computational Linguistics, Proceedings of the Conference, 18-22 August 2008, Manchester, UK*, pages 593–600, 2008.

[79] Amihai Motro. Intensional answers to database queries. *IEEE Trans. Knowl. Data Eng.*, 6(3):444–454, 1994.

[80] Amihai Motro. Cooperative database systems. *Encyclopedia of Library and Information Science*, 66:79–97, 2000.

[81] Hassina Mouloudi. *Personalisation of OLAP queries and visualisations under constraints (in french).* PhD thesis, Université François Rabelais Tours, 2007.

[82] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.

[83] Elsa Negre. *Collaborative exploration of data cubes (in french).* PhD thesis, Université François Rabelais Tours, 2009.

[84] Patrick E. O'Neil, Elizabeth J. O'Neil, Xuedong Chen, and Stephen Revilak. The Star Schema Benchmark and Augmented Fact Table Indexing. In *Performance Evaluation and Benchmarking, First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009, Revised Selected Papers*, pages 237–252, 2009.

[85] Carlos Ordonez and Zhibo Chen. Evaluating Statistical Tests on OLAP Cubes to Compare Degree of Disease. *IEEE Transactions on Information Technology in Biomedicine*, 13(5):756–765, 2009.

[86] Torben Bach Pedersen. How Is BI Used in Industry?: Report from a Knowledge Exchange Network. In *Data Warehousing and Knowledge Discovery, 6th International Conference, DaWaK 2004, Zaragoza, Spain, September 1-3, 2004, Proceedings*, pages 179–188, 2004.

[87] Veronika Peralta, Dimitre Kostadinov, and Mokrane Bouzeghoub. Apmd-workbench: A benchmark for query personalization. In *Workshop on Contextual Information Access, Seeking and Retrieval Evaluation (CIRSE)*, 2009.

[88] Meikel Pöss, Raghunath Othayoth Nambiar, and David Walrath. Why You Should Run TPC-DS: A Workload Analysis. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 1138–1149, 2007.

[89] Stefano Rizzi. OLAP preferences: a research agenda. In *DOLAP 2007, ACM 10th International Workshop on Data Warehousing and OLAP, Lisbon, Portugal, November 9, 2007, Proceedings*, pages 99–100, 2007.

[90] Stefano Rizzi. New Frontiers in Business Intelligence: Distribution and Personalization. In *Advances in Databases and Information Systems - 14th East European Conference, ADBIS 2010, Novi Sad, Serbia, September 20-24, 2010. Proceedings*, pages 23–30, 2010.

[91] Oscar Romero and Alberto Abelló. On the Need of a Reference Algebra for OLAP. In *Data Warehousing and Knowledge Discovery, 9th International Conference, DaWaK 2007, Regensburg, Germany, September 3-7, 2007, Proceedings*, pages 99–110, 2007.

[92] Oscar Romero, Patrick Marcel, Alberto Abelló, Verónika Peralta, and Ladjel Bellatreche. Describing Analytical Sessions Using a Multidimensional Algebra. In *Data Warehousing and Knowledge Discovery - 13th International Conference, DaWaK 2011, Toulouse, France, August 29-September 2,2011. Proceedings*, pages 224–239, 2011.

[93] Carsten Sapia. PROMISE: Predicting Query Behavior to Enable Predictive Caching Strategies for OLAP Systems. In *Data Warehousing and Knowledge Discovery, Second International Conference, DaWaK 2000, London, UK, September 4-6, 2000, Proceedings*, pages 224–233, 2000.

[94] Sunita Sarawagi. Explaining Differences in Multidimensional Aggregates. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 42–53, 1999.

[95] Sunita Sarawagi. User-Adaptive Exploration of Multidimensional Data. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 307–316, 2000.

[96] Sunita Sarawagi, Rakesh Agrawal, and Nimrod Megiddo. Discovery-Driven Exploration of OLAP Data Cubes. In *Advances in Database Technology - EDBT'98, 6th International Conference on Extending Database Technology, Valencia, Spain, March 23-27, 1998, Proceedings*, pages 168–182, 1998.

[97] Gayatri Sathe and Sunita Sarawagi. Intelligent Rollups in Multidimensional OLAP Data. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pages 531–540, 2001.

[98] Markus Schneider, Gottfried Vossen, and Esteban Zimányi. Data Warehousing: from Occasional OLAP to Real-time Business Intelligence (Dagstuhl Seminar 11361). *Dagstuhl Reports*, 1(9):1–25, 2011.

[99] Alkis Simitsis, Georgia Koutrika, and Yannis E. Ioannidis. Précis: from unstructured keywords as queries to structured databases as answers. *VLDB J.*, 17(1):117–149, 2008.

[100] Temple Smith and Michael Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[101] K. Stefanidis, M. Drosou, and E. Pitoura. "You May Also Like" results in relational databases. In *Proceedings International Workshop on Personalized Access, Profile Management and Context Awareness: Databases*, Lyon, France, 2009.

[102] Kostas Stefanidis, Georgia Koutrika, and Evaggelia Pitoura. A survey on representation, composition and application of preferences in database systems. *ACM Trans. Database Syst.*, 36(3):19, 2011.

[103] Raphaël Thollot, Nicolas Kuchmann-Beauger, and Marie-Aude Aufaure. Semantics and usage statistics for multi-dimensional query expansion. In Sang goo Lee, Zhiyong Peng, Xiaofang Zhou, Yang-Sae Moon, Rainer Unland, and Jaesoo Yoo, editors, *DASFAA (2)*, volume 7239 of *Lecture Notes in Computer Science*, pages 250–260. Springer, 2012.

[104] The Transaction Processing Performance Council (TPC). Tpc benchmark ds (tpc-ds): The new decision support benchmark standard. http://www.tpc.org/tpcds/, April 2012.

[105] Juan Trujillo and Alejandro Maté. Business intelligence 2.0: A general overview. In Marie-Aude Aufaure, Esteban Zimányi, Wil Aalst, John Mylopoulos, Michael Rosemann, Michael J. Shaw, and Clemens Szyperski, editors, *Business Intelligence*, volume 96 of *Lecture Notes in Business Information Processing*, pages 98–116. Springer Berlin Heidelberg, 2012.

[106] Jovan Varga. Multidimensional query recommendation. Master's thesis, Universitat Polytecnica de Catalunya, 2011.

[107] Panos Vassiliadis and Spiros Skiadopoulos. Modelling and Optimisation Issues for Multidimensional Databases. In *Advanced Information Systems Engineering, 12th International Conference CAiSE 2000, Stockholm, Sweden, June 5-9, 2000, Proceedings*, pages 482–497, 2000.

[108] Adriano Veloso, Humberto Mossri de Almeida, Marcos André Gonçalves, and Wagner Meira Jr. Learning to rank at query-time using association rules. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2008, Singapore, July 20-24, 2008*, pages 267–274, 2008.

[109] Xiaoyan Yang, Cecilia M. Procopiuc, and Divesh Srivastava. Recommending Join Queries via Query Log Analysis. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 964–975, 2009.

[110] Qingsong Yao, Aijun An, and Xiangji Huang. Finding and Analyzing Database User Sessions. In *Database Systems for Advanced Applications, 10th International Conference, DASFAA 2005, Beijing, China, April 17-20, 2005, Proceedings*, pages 851–862, 2005.