Computation of Extended Answers to Relational Database Queries

Aurélien Moreau

Research Master Thesis

Research Master Thesis

Computation of Extended Answers to Relational Database Queries

Author:
Aurélien Moreau

Supervisors:
Olivier Pivert
Grégory Smits
IRISA – Shaman (ex-Pilgrim)

ENSSAT
LANNION

Abstract

This Master thesis is divided into two main parts: a state of the art and the work achieved during the internship. The bibliographic part highlights several aspects related to database browsing and recommendation systems. Assisting the user with keyword search or cooperative answers has been a recurring issue in the world of databases. Providing users with recommendations is of major importance in electronic commerce. Inspired by both recommendation systems and cooperative answers, the Shaman research team has outlined a new approach to compute similar answers. This approach based on fuzzy associations is at the heart of the Master's degree internship. The main objectives include giving more insight as to why some answers are returned and automatically finding similarity criteria for a given object amongst others, which are further detailed in this report.

Résumé

Ce mémoire de Master est divisé en deux parties : un état de l'art et un compte rendu du travail réalisé pendant le stage. La partie bibliographique met en valeur certains aspects liés à la recherche dans les bases de données et aux systèmes de recommandation. Aider l'utilisateur avec la recherche par mots clés et les réponses coopératives a été un problème récurrent dans le monde des bases de données. Proposer aux utilisateurs des recommandations est d'importance vitale dans le commerce électronique. Inspiré par les systèmes de recommandation et les réponses coopératives, l'équipe Shaman a décrit une nouvelle approche pour calculer des réponses similaires. Cette approche basée sur les associations floues est au coeur du stage de Master. Les objectifs principaux comprennent entre autres l'explications des réponses retournées, ainsi que la découverte automatisée de critères de similarité pertinents pour un objet donné.

Contents

List of Figures

List of Tables

Introduction

Databases have become an essential tool in most of our everyday lives. We simply may not realize this. Just about every information system requires its data to be organized according to a model fitting reality, be it a student's schedule at university, or any e-commerce company store or any search engine. In this paper we will consider relational databases, i.e. databases based on a relational model. Retrieving information from such databases usually requires knowledge of a querying language such as SQL. Not only users have to query for data, but the data returned may not always be what the user was looking for. In order to make database browsing available to more users, several forms of keyword search have been implemented over the years. Also, mechanisms have been introduced to add more relevant information to the returned answers. These attempts to make databases more user-friendly have led to the development of cooperative answering [14] which consists in taking into account the preferences of the user and eventually returning relaxed answers for instance.

When browsing for a particular item on the Internet, additional answers are often returned and qualified as "recommendations". These are also called similar answers, as opposed to direct answers to the query. Providing both direct and similar answers constitutes enriched answers, a form of cooperative answering. Recommendation systems have become very popular inasmuch as they can promote content probably unknown to the user and yet of interest to them. Just like the Internet, as the needs of the users change, so do the functionalities of recommender systems: if they simply needed attribute values and/or ratings before, now they can also use localization and social links to compute suggestions. And this is all the more true when people tend to always have digital devices around them, be it their mobile phone, a tablet or a laptop. Many of us are connected at all times, browsing data or information, and are thus prey to recommender systems.

One of the notions at the heart of recommender systems is similarity search. While very dense in terms of applicability – recommending pictures will be very different from recommending books – the main issue inherent to similarity search stems from the complexity of the considered objects. In some cases similarity based on values is easy to compute. However defining similarity criteria can be problematic when they are not completely obvious as a certain degree of subjectivity may then interfere. The similarity of values for attributes is often used if an adequate metric is available. Subsets of similar items may be delimited considering the typical properties of a given item. It is then possible to look for other items also having these typical properties. In order to facilitate the discovery of information and distinguish typical elements, an ulterior motive would then be to single out atypical properties of items in a set of typically similar items.

Linking relational databases and recommendation systems, Stefanidis *et al.* [25] proposed recommendations in databases denoted as "You May Also Like" or YMAL answers. Their computations have been classified into several categories – current-state, history-based, and external sources

– based on the recommendation approach considered, the first of which only uses the content of the database and the results of a query, looking for patterns in these results. The second is inspired by classic recommender systems, using the history of items browsed. The last one focuses on information gathered on the Web, in sources different from the considered database. These particular types of answers should not be confused with preference queries such as the top-k – finding the k most fitting answers – and fuzzy queries – using graduality to rank the answers – for instance. These mainly take into account the graduality of the satisfaction in accordance with user preferences. In other words, they are still considered as "direct answers" – however relaxed they may actually be – and not as "similar answers". Yet they can still be qualified as cooperative answers.

This report presents a state of the art regarding a few aspects of relational databases as well as recommendation systems. This state of the art was drafted as a preparation for the internship, revolving around the computation of extended answers based on fuzzy associations between entities. Defining similarity criteria is the main challenge as the notion of similarity is the cornerstone of this approach. Computing both typical and atypical attributes in a set of similar items is another way of providing relevant information to the user. Most of the examples used throughout this report are based on movies, and so is the original prototype providing association-based recommendations. However every concept can be generalized to any field as long as there is enough data and an expert to govern it.

Chapter 1

State of the Art

This chapter provides a state of the art covering topics at the crossroads between databases and information retrieval, such as keyword search in databases and recommendation systems. Our issue is to return similar answers that do not match the query but that would be of interest to the user. In Section 1.1 we highlight several aspects of querying assistance in databases, and keyword search in particular. Section 1.2 presents an overview on recommendation systems and how they can be used with databases. Section 1.3 highlights the critical differences between related work and our objectives. In Section 1.4 we present an approach based on fuzzy associations which has been further researched during the Master's degree internship with the Shaman research team and finally we draw a conclusion of this chapter in Section 1.5.

1.1 Assisting Users Querying Databases

Querying databases can be quite difficult for various reasons. To browse a database you usually need a querying language, have a precise idea of what you are looking for and where exactly in the database this information is, assuming knowledge of the schema of the database. Thankfully a few mechanisms have been introduced to facilitate the use of databases, making them more accessible to users who lack such knowledge. Keyword browsing is one of them. While being too vague when querying a database may lead to plethoric answers, being too precise may lead to no answer at all. An alternative to keyword search is to browse data step by step, called faceted search. Otherwise, similarity search has enabled the discovery of content potentially interesting to the user.

1.1.1 Keyword Search Approaches with Joining Networks

With a few keywords it is possible to browse through the content of a database and return tuples satisfying the keywords, and the user does not even need to know the schema of the database. Originally, this consisted in simply returning the tuples matching the keywords exactly, without the possibility to join tables to provide more complete results. Being too lax would return countless results without the possibility to rank them in any way, thus having a hard time finding the information desired. The objective of the following systems is to better cover a keyword query, with new mechanisms.

BANKS [5] aims to retrieve the results and is also capable of ranking them using prestige and proximity. BANKS starts off with modelling the database as a directed graph. Tuples thus

become nodes, and the links between foreign keys and primary keys correspond to directed edges between tuples. Then the keywords are compared to data and meta-data to cover the whole of the database. It has been tested – and can be tested[1] – over versions of DBLP[2] and IMDB[3]. To find answers of interest, the idea is to determine a node in the graph connected to every node related to the keywords. The resulting sub-graph is called a tree whose root is the aforementioned node connecting the other nodes. The links between nodes have different weights to infer the notion of prestige: in a movies database the link between the tables *Movies* and *Producers* is weaker than the link between the tables *Movies* and *Directors* – assuming we are more interested in the directors than in the producers. The weaker the link between the tables, the heavier the weight between the nodes is. The relevance of the answers is inversely proportional to the total weights of the links. Obviously these weights are computed beforehand by the programmer. Also, node weights are introduced. The idea is that the more a node is linked in the graph, the more important it is. That way, the more links an *Actor* node has with *Movies* nodes, the more important it is. After having computed the answer trees, they are ranked according to the total weight of their nodes and relations. One drawback to this approach is that computing the graph and comparing all the nodes requires a lot of memory space. While this system has been tested, it is nonetheless always assumed that there would be no shortcomings on memory space.

DBXplorer [4] is a system for keyword-based search over relational databases, providing all the rows containing *all* the given keywords. In other words, no rows only "partially" adequate will be returned. Reducing the need of space and improving the relevance of search results is another goal of the authors. Exploiting the schema and the content of the database are both necessary to obtain the expected results. Full text search (FTS) is another requirement for extensive search over the string attributes of the database. To do that the database needs to be prepared for keyword search i.e. identifying the tables and creating other tables to enable keyword search, called auxiliary and symbol tables. This is the publishing part. Then we can submit a query that will result in the joining of tables to eventually return only the rows containing all keywords.

DISCOVER [16] could be qualified as an "add-on" for relational databases, implementing keyword search. As a consequence there is no need for huge quantities of memory space. It exploits the schema graph to figure out how the relations within the database work. It operates in two steps: first the "candidate network generator" and then the "plan generator". Let us consider a database with n relations, and that every relation R_i has m_i attributes. From a set of keywords k_1, ..., k_p sent to the master index, we get "basic tuple sets" $S_{(i,j)}$ with $i = 1, ..., n$ and $j = 1, ..., p$ such that $S_{(i,j)}$ contains every tuple of relation R_i with the keyword k_j. The tuple set post-processor uses these tuple sets to produce sub-sets with only certain keywords in them. They are then sent to the candidate network generator. With help from the schema of the database, this generator will provide candidate networks which will finally be evaluated with the plan generator. While there is no redundancy in the results, they are not ranked in any particular order. When testing this approach with the TPC-H database[4], the authors realized that because this database lacked diversity – not enough different keywords – their approach was not efficient. The system DBXplorer [4] was also tested on this database and returned satisfactory results, as the implementation of the master index was very good and so resulted in fast execution times. However, with DISCOVER,

[1] http://www.cse.iitb.ac.in/banks/banks-demo/
[2] http://www.informatik.uni-trier.de/ ley/db/
[3] http://www.imdb.com/
[4] http://www.tpc.org/tpch/

keywords do not have to exactly match attribute values.

The common idea to these approaches – all three published in the same year – was to exploit the relationships between foreign and primary keys. In other words, to use the links of the schema to navigate efficiently through the database and then deploy networks or model graphs to represent the new links and the potential rows satisfying the keywords.

1.1.2 Similarity Search

Keyword search may not always be the most efficient approach when looking for information. While it is direct, it can also sometimes miss the point, especially when facing plethoric answers. Being able to navigate through hierarchical structures can help a user find exactly what they may desire, using classifications. One example would be the categories or departments for items on an e-commerce website. This is faceted search. This browsing technique is heavily involved in the CiteSeerX[5] project. While this approach has been largely described in [34, 26] we will not dwell on it any further as it is rather out of scope.

The relationships between documents and related objects such as authors, products or persons have been acknowledged and studied in [7], introducing "object finder" queries. From a set of keywords, these queries aim to return the top K objects that fit the keywords. The main issues addressed here are to determine how exactly objects can match keywords, and how to rank these objects. Ranking the objects is the main challenge as a few papers had already tackled the issue of keyword browsing as shown in Sub-section 1.1.1. Full Text Search has been implemented to analyze every inch of the documents. For example consider a client database with reviews for products. These reviews may contain interesting keywords. A product with several reviews containing a selected keyword will be considered important. A complication may be linked to the semantics of the reviews: if users comment negatively on a feature then the product will still be considered important in regard to that feature. When browsing laptops, should we be interested in laptops with a "good battery life", comment containing the string "not a good battery life" will also be taken into account. When the user stumbles on said reviews they will simply discard the item, however simply not returning this item would be better. One drawback is that this approach may be very limited when it is applied to huge databases, as we would be facing too many potential answers and thus having trouble ranking them.

The notion of meta-path was introduced in [28] as a new paradigm to for similarity search. Let us consider another cinematographic example with movies, directors and actors. With such paths co-actors would be linked by a path "actor-movie-actor" or movies directed by a same director by a path "movie-director-movie". As simple as these paths are, it will obviously get as complicated as "actor-movie-director-movie-actor" – actors starring in movies of the same director – or "actor-movie-actor-movie-director-movie-actor-movie-actor" – actors whose co-actors have starred in movies directed by the same director – if not more complicated than that. Thus the authors introduced a new similarity measure based on meta-paths, called PathSim, where the idea is that the more links objects have between each other, the more similar they are. This approach can be used on "heterogeneous information networks", so it is not restricted to the world of relational databases. An interesting example is that the authors browsed for similar pictures on the Flickr network[6] which is not at all based on a relational model. Thanks to the meta-data attached to

[5] http://csxstatic.ist.psu.edu/about

[6] http://flickr.com/

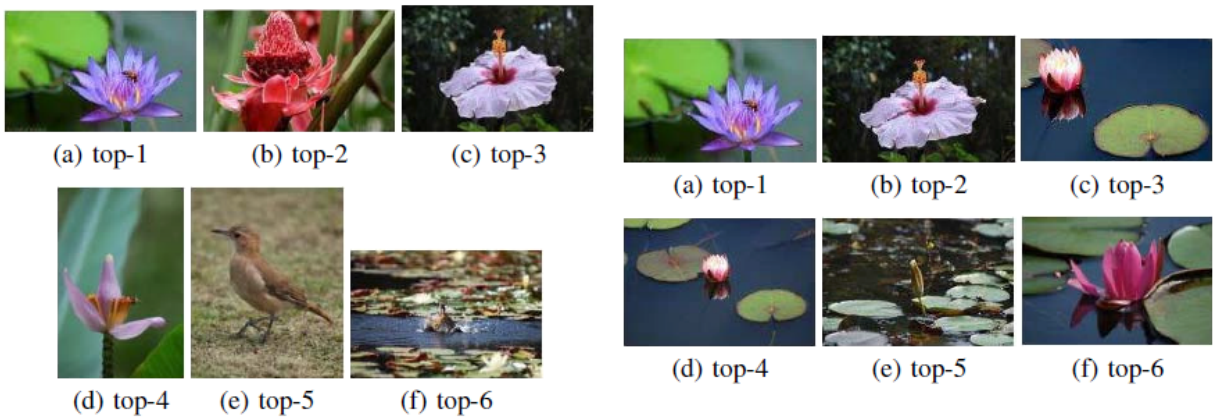

Figure 1.1: Similar pictures with two different meta-paths. On the left, pictures with the path picture-tag-picture and on the right pictures with the path picture-tag-picture-group-picture-tag-picture. From [28]

pictures, they computed two meta-paths: picture-tag-picture – featuring pictures with the same tags – and picture-tag-picture-group-picture-tag-picture – featuring pictures which have the same tag as pictures in the same group. Interestingly enough the second meta-path returned more accurate results as shown in Figure 1.1. But this does not mean that the longer the meta-path, the more accurate the results will be. The authors proved that the behaviour of PathSim with infinite meta-paths was limited.

We have seen that browsing data in databases can be done with help from both the schema and the content. However browsing databases with keywords only does not always return accurate results. On the fringe of databases, a new type of system made a break-through a few years ago: recommendation systems. They were popularized by a competition[7] that rewarded the best algorithm returning the most accurate recommendations for movies on a dataset of more than 100 million movie ratings. Starting with an item – or several of them – and then returning similar items is another way to return extended answers: let us discover the mechanisms behind them in the following section.

1.2 Recommendation Systems-Related Approaches

Nowadays many web services offer suggestions to their users: be it an e-commerce website, a video browsing website, an online newspaper, or advertisements as a whole. This is possible with recommendation systems, which have become very popular over the past few years.

1.2.1 An Overview of Recommendation Systems

Recommendation systems can be categorized into two main groups [22, 2]: content-based and collaborative filtering systems. Combining the two makes up hybrid recommendation systems. As time passes, more and more different ways to use recommendation systems have come up, thanks to the emergence of the Web 2.0 and then of the Web 3.0.

[7] http://www.netflixprize.com/

Original Recommender Systems

Content-based recommendation systems use the similarities of the attributes between items. For each and every item, a profile containing all properties must be created. If we consider movies, the Internet Movie Database offers most of the data available related to movies such as the cast, the director, the year of release and the genres for instance. All these are important properties interesting to compute recommendations. The idea is to recommend items similar to the ones liked by the user: with the same actors, or the same directors, or the same genres. As it is often keywords that are associated with attribute values, the importance of these keywords needs to be measured depending on where they come from. The term frequency/inverse document frequency (TF.IDF) is one of the most used measures to compute the specific features of documents. However when considering media content such as music samples or pictures it becomes more complicated to use this approach as besides some meta-data there are not always many usable attributes associated with these files. As a consequence, even though there may be some data available, it is not necessarily enough to compute appropriate recommendations. Following an analysis of the items, the users need to be analyzed too, based on the items they browsed – and maybe rated – in the past. Preferences and tastes are inferred and recommendations can then be computed. It may take a few ratings to get appropriate recommendations to a new user.

Collaborative filtering systems use relations between items and users, recommending items "liked" by similar users. A form of feedback is necessary for this kind of recommendation. Be it a rating, or simply having browsed the item, the users need to interact with items in order for this system to acquire data. Similar behaviours are then detected (having browsed the same items, or given similar ratings to some items) and users can then be considered similar. From then on it is possible to recommend to a user items browsed or liked by similar users. The obvious issue to this system is known as the cold start, when there are no data to provide any recommendations. Having a new user or a new item are two adjacent problems.

To palliate the drawbacks of each of these two systems, it is possible to combine them into a hybrid system. To do this, it is possible to implement the two systems separately and then combine the recommendations, or mainly implement one of the two systems and add a few characteristics of the other, or create an original system involving methods from both systems. For example, with new items it would be possible to rely on content-based methods only for a while until there are enough data to use collaborative filtering algorithms.

New Uses of Recommendation Systems

Every time there is a new Web revolution, uses change. With the Web 2.0, social networks emerged fairly fast and an obvious new way of providing recommendations came up: people that are simply "close" on social networks may have similar tastes. New factors had to be taken into account: trust and reputation. Trust in users – or "friends" – and trust in items – through feedback, ratings, or demonstration videos – raise the credibility of recommendations. Bobadilla *et al.* wrote a survey in [6] in which they separated trust and reputation mechanisms under three categories: explicit trust systems, implicit trust systems and memory based trust. Beside these new factors, more classic approaches based on content-based filtering tend to have new issues with social information: overspecialization, attribute generation and feedback gathering. Overspecialization is due to recommender systems often providing items too similar to those the users already have. The challenge here is to provide recommendations for items that would go well with items already obtained. At-

tribute generation – required to compute similarities on values – to perform content differentiation may be difficult when considering media content such as pictures, movies and music. The most practical data available will be meta-data, but it is not always reliable and efficient to compute recommendations. Feedback gathering is more complicated on social networks because users may "rate" items very casually by simply giving their opinion for all the world to see, regardless of any feedback mechanism. In other words getting the rating to compute recommendations is another challenge. Some may simply state that they "like" or "dislike" it, or not rate it in any way at all.

Other challenges include recommending to groups of users – such as recommending a movie to a group of four going to the theatre – and explaining recommendations – justifying results tends to increase confidence from the users. The new advance in technology is the Internet of Things, or the Web 3.0. Now recommendation systems have the possibility to take into account the localization of the user as well as the context due to the emergence of smartphones: they can recommend places that are near the user such as restaurants.

1.2.2 When Recommendation Systems and Databases Meet

The concept of recommending items is based on browsing and analyzing available content. But most recommendation systems do not make use of any existing data structures, which is why using both databases and recommending systems together enables the use of the schema of the database to provide more interesting recommendations.

Recommendations in databases are different from those on the Web inasmuch as users browse the Web and databases differently. While on the Web it is easy to browse items, with a database you have to query for them. And this is not necessarily an easy task especially when you have to look across several tables. The essence of the QueRIE system described in [8] is different from usual database browsing systems as the authors are not interested in computing the wishes of users directly but strive for more insight on the means: in other words they are trying to provide new queries – new means – for the users. While tuples can be considered unique – primary keys guarantee the unicity – the queries to return some tuples are certainly not. That is why analyzing the tuples is not the point here, as they can be returned in several ways. QueRIE recommends queries that would provide interesting results to the user. As users may have different interests, the notion of session is introduced. That way, a user interested in movies and in sailing will not come across queries including both topics at once *i.e.* movies about sailing for instance. For every subject – meaning every time the user uses the system, a new session is created. In order to recommend queries, the system will compare the queries formulated by the user – in one given session – with queries in the sessions of other users. Users whose sessions contain the exact same queries – or similar ones more likely – would then provide new queries for the current user. Let us note that the queries recommended could very well return the same results as those already retrieved by the user. There are two elements taken into account to recommend queries: the results returned by queries, and the fragments of the queries. Inspired by collaborative filtering recommendations, this system needs a set of users having interacted with the database beforehand to provide query recommendations. Let us consider a user with its current session, about movies. Now the user has queried for movies with the title containing the string "Harry Potter". Let us assume that another user has already queried for movies in which the actor Daniel Radcliffe starred – Daniel Radcliffe starred as "Harry Potter" in 8 movies. As both queries should return a certain number of movies in common, the users could be considered similar. So the other queries formulated by the second user may then be suggested to the first one.

However recommendation systems tend to lack the possibility of personalization. Some can be unmodifiable, completely set in the code. Others offer no flexibility such as recommending items based on two different users' wishes: two users with different tastes wishing to watch a movie together will not be satisfied with recommendations fitting only one user. And then there are systems limiting the number of parameters on which recommendations can be computed e.g. user ratings for movies or subjects in research. FlexRecs [18] is a framework promoting customizable recommendations. Created to answer a need for some guidance in the selection of courses in college, it can recommend courses based on a user's past class grades and ratings, which can then be compared to other students' to formulate recommendations. This is done with the help of common operators such as *select*, *project*, *join*, and with more unusual but nonetheless appropriate operators such as *extend*, *recommend* and *blend*. Recommendations are computed from the similarity between values. When the values are strings, the Jaccard similarity is applied. When the values are numerical, distances are used. Conditional probabilities and extended values can be used too.

Flexible recommendations in databases were proposed in [1] and [3] with the Recommendation Query Language (RQL). This language allowed knowledgable people to compute recommendations à la SQL. Known issues included computation time because RQL queries translated into SQL tend to be rather complex.

1.2.3 You May Also Like Queries

You May Also Like results were defined by Stefanidis *et al.* [25] as tuple recommendations to queries on relational databases. The computations for these results have been categorized in several approaches: current-state, history-based and external sources. We will describe the principle of each one of these. Unlike FlexRecs [18], the authors rely on classical relational algebra operators. The principles outlined in [25] mainly remain general principles: they have not been implemented extensively, as the authors focused on only one of the approaches.

This approach only considers a query q and its result $R(q)$. Several analyses are then possible: a *local analysis* of the characteristics of $R(q)$ or a *global analysis* of the characteristics of the database. These two analyses can use both the content and the schema of $R(q)$ – *local analysis* – or of the database – *global analysis*. The two can also be combined to create hybrid analyses. The *local analysis* consists in analyzing the recurring patterns in the tuples of the result $R(q)$, and then offer items that fit in these patterns and do not belong to $R(q)$. Let us consider a movie database, and let us look for movies in which Daniel Radcliffe starred. As Daniel Radcliffe has often starred as a wizard, recommending movies with witches and wizards would make sense. The *global analysis* uses the characteristics of the database as a whole such as the tables, the attributes, and the links between tables. When using the content of the database, if we look for Daniel Radcliffe movies, we could recommend Rupert Grint movies since these two often starred together. These approaches are the most similar to classic recommending systems. In other words there is a need for previous interactions between users and the database. They can be put into two categories: query-based results – content-based recommendations, analyzing the similarity between queries – or user-based results – collaborative filtering recommendations, analyzing the similarity between users. Again they can be combined to create hybrid approaches. The idea is to use any other semantic resources such as the Web to find related answers. This is an open approach, here to welcome everything that does not fit in any of the other two. For example the main idea would be to use information that is not usually available when browsing movies attributes. One example would be the link between father Jon Voight and daughter Angelina Jolie: such a relationship is not featured in any classic

movie database schema.

This notion of You May Also Like answer was extended in [9] by two authors of [25] into the YmalDB system. They defined the notion of *faSets* as sets of interesting values. How relevant and interesting they are is computed based on their frequency in both the original query and in the database content. Computing such frequencies on the fly for each query is not exactly feasible, so they used representative *faSets* and summaries to keep track of the average frequencies in the database. For example, they looked for the data related to movies directed by *Martin Scorsese*, such as the years, the genres and the countries related to his movies. It appears that *Martin Scorsese* often directed movies of the genre *Biography*, so the *faSet* highlighting elements related to *Biography* movies will be presented to the user. This does not necessarily include *Biography* movies only, but also actors who often starred in *Biography* movies.

1.3 Discussion

Keyword browsing [4, 5, 16] introduced interesting notions – in particular joining networks and the graph modelling of a database – although the aim here was not to recommend any new items but simply to fulfill keyword queries in a more sophisticated way considering keywords that may refer to different tables of the database. Others [8] were not exactly interested in retrieving similar items directly but similar queries. Furthermore they did not make any use of foreign and primary keys constraints.

PathSim [28] focuses on the number of meta-paths between objects. This is rather new and we will see that the approach described in Section 1.4 is based on a somewhat similar concept. FlexRecs [18], like most recommending systems, bases its recommendations on the similarity between values. Even though the framework proposed is quite innovative – allowing the user to personalize recommendations effectively – the means to compute these recommendations are not so original. RQL [1, 3] is a Recommendation Query Language capable of computing suggestions for any user, provided that they have notions of any programming language in the first place. While the flexibility given here is inspiring, the complexity of the queries make it very difficult to use because of fairly long execution times. Recommender systems as a whole tend to either use feedback from the users with collaborative filtering, or compare the items' attributes directly with content-based approaches. Doing that, they mostly neglect another important part of databases: the schema.

In [11] the authors presented TupleRecommender, a system capable of giving recommendations in relational databases from a keyword query. The keyword search system is based on DBXplorer [4] and DISCOVER [16], while the recommendation approach simply compares tuples pairwise. In other words they combine the schema-level analysis of the database to look for tuples and then use content-based approaches on the tuples to provide recommendations. For instance, they started looking for *James Cameron*, *Zoe Saldana*. It turns out that the former is the director and the latter the main actor of the Avatar movie. The first recommendation presented is Alien, directed by the same director and the two movies are of the same genre, which is science fiction.

Most approaches related to keyword search and recommendation systems presented here are summed up in Table 1.1. Some fall into several categories considering that they are hybrid approaches implementing several techniques. The main comparison between systems is made between the use of the schema and/or the content of the database. While using the content usually guarantees convincing results, this comes at the expensive price of space and time. On the other hand

	Keyword Search	Recommender Systems
Schema Level	DISCOVER DBXplorer TupleRecommender ObjectRank	PathSim QueRIE YmalDB
Tuple Level	ObjectRank BANKS	TupleRecommender FlexRecs RQL YmalDB

Table 1.1: Classification of the different approaches mentioned

relying solely on the schema may lead to poor results should it be unexploitable. The approach whose principle is described in the next section clearly belongs to the category of methods that the authors of [25] call "current-state approaches", but its originality lies in the exploitation of the notion of association between entities from different tables, which is not explicitly mentioned by the authors of [25].

1.4 An Association-Based Fuzzy Approach

In [21], members of the Shaman research team have proposed a new approach to compute similar objects in relational databases. The use of fuzzy logic along with foreign keys makes it possible to discover and rank objects similar to one another. It is categorized as a "current-state approach" in the sense of [25].

1.4.1 Main Principle

The objective is to give the user, in addition to direct answers to their query, items that would be of interest to them. The idea is to look for items similar to those directly returned, taking into account associations between some items from the database and the items directly returned. For instance, in a cinematographic database, actors would be considered similar to a given actor should they have

- played mainly for the same directors as the given actor,
- played mainly in movies of the same genres as the given actor,
- a set of most frequent co-actors that is similar to the given actor's.

From then on the principle is to compute the multisets of the comparison objects (directors, genres, or co-actors) of the given actor, and those of other actors in the database. Then to compute the fuzzy sets of *typical* values for the multisets. And finally to compute a degree of matching between the above fuzzy sets. This degree is then used to rank the results.

Let us consider an element x_i from a multiset E containing n elements. Its frequency will be denoted by f_i such as:

$$f_i = \frac{n_i}{n} \tag{1.1}$$

where n_i is the number of times x_i can be counted in E.

Now we desire to compute the fuzzy set T so that the typicality μ_T of an element x_i in a multiset E can be simply defined as

$$\mu_T(x_i) = f_i. \tag{1.2}$$

A more flexible way to define typicality is to introduce the fuzzy quantifier *most*, so that:

$$\mu_T(x_i) = \mu_{most}(f_i) \tag{1.3}$$

where *most* has the behaviour depicted in Figure 1.2. Frequencies below δ are considered not typical at all, above γ they are considered totally typical and in between the typicality increases linearly.

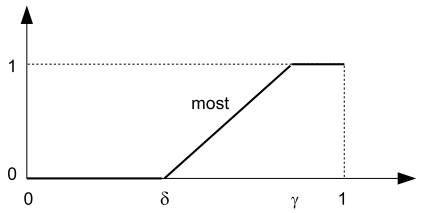

Figure 1.2: The fuzzy quantifier *most*, from [21]

Once the fuzzy sets of typical values for the multisets have been computed, they have to be compared with the help of fuzzy matching operators. A few have been moved forward such as the Jaccard index:

$$\mu_{matches_1}(E_1,\ E_2) = \frac{\sum_{x \in U} \min(\mu_{T_1}(x),\ \mu_{T_2}(x))}{\sum_{x \in U} \max(\mu_{T_1}(x),\ \mu_{T_2}(x))} \tag{1.4}$$

where U denotes the underlying domain of E_1 and E_2, or if we are interested in more drastic measures computing how every value is as typical in E_1 as in E_2:

$$\mu_{matches_2}(E_1,\ E_2) = \inf_{x \in U}\ 1 - |\mu_{T_1}(x) - \mu_{T_2}(x)| \tag{1.5}$$

Other matching measures were proposed, and other ones could certainly be proposed. Let us consider a bibliographic database and assume that two authors are considered similar if the typical sets of journals in which they publish are similar. Let us consider the typical sets:

$$T_1 = \{0.2/\text{PVLDB},\ 0.3/\text{TKDE},\ 0.6/\text{JIIS},\ 0.6/\text{DKE}\}$$

and

$$T_2 = \{0.3/\text{DKE},\ 0.4/\text{TODS},\ 0.6/\text{PVLDB},\ 0.8/\text{IJIS}\}.$$

The similarity degrees obtained using the previous measures are:

- with Formula (1.4): $\mu_{matches}(E_1,\ E_2) = \frac{0.5}{3.3} \approx 0.15$
- with Formula (1.5): $\mu_{matches}(E_1,\ E_2) = inf(0.6,\ 0.7,\ 0.4,\ 0.7,\ 0.6,\ 0.2) = 0.2.$

Although these results are fairly close, it should be noted that other matching measures can compute very different values, with a similarity up to at least 0.45 with this example. The approach has been implemented over the Internet Movie Database[8]. Figure 1.3 illustrates some of the tables used in this work, their attributes, cardinalities and relations. Two criteria have been considered:

[8] http://www.imdb.com/

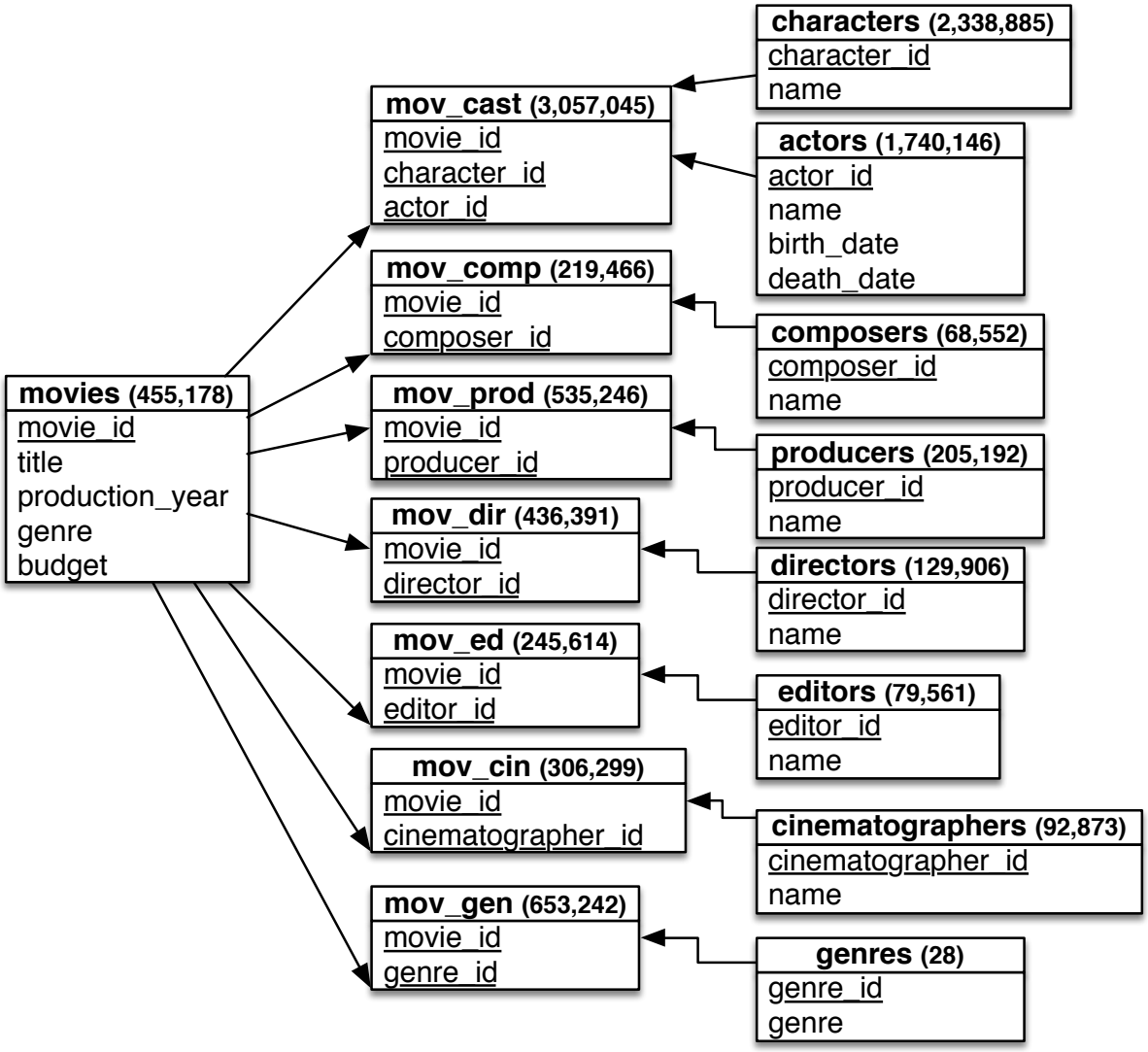

Figure 1.3: Part of the schema of IMDb

having worked with the same directors, and having played with the same other actors makes two actors similar. They can be used separately or together.

In order to evaluate the relevance of the answers produced from a target actor, they were then submitted to users who classified the results as either "relevant", "irrelevant", or "I don't know". According to the average relevance the best precision would be based on the Jaccard index. Although a rather high number of people answered "I don't know" to several entries, it was somehow expected as the point of recommending systems is to make people discover new items. However there are no explanations as to "why" exactly actors are recommended, which can certainly be a little disorienting.

1.4.2 Future Work

Following these encouraging experiments, there are several issues which will require our attention. We need to compare more matching measures to compute the similarity between fuzzy typical multisets. We wish to figure out how it could be possible to determine automatically which criteria would be relevant to compute the similarity between given objects. A few researchers [27] have already ventured in this field with the example of e-commerce.

Understanding why exactly some answers are returned and evaluating the relevance of returned answers are the most important points. Georgia Koutrika, one of the authors of FlexRecs [18] wrote papers of interest on this topic [19, 24]. Indeed, knowing why exactly an actor is deemed similar to another would probably encourage people to know more about this recommended actor if the explanation satisfies them.

Optimizing the computation of results with adaptive pruning strategies is essential to reduce execution times, especially when dealing with large databases such as IMDB. Finally we shall consider pairing this approach with a classic one where the similarity between objects is computed based on the values of attributes.

1.5 Conclusion

This bibliographic study highlighted previous works in the fields of databases and recommender systems. We have presented several mechanisms such as keyword browsing among others to assist users in browsing relational databases. Most of these strove to better cover an initial query and did not inquire similar objects, but had interesting approaches to ponder on. Then we presented a few recommending systems applied to databases, most of which were rather classical. Figuring out how to automatically compute similarity criteria for a given object is the major challenge, along with determining why some answers are returned. The association-based fuzzy approach brought forward by the Shaman research team needs to be refined, and that is exactly the purpose of this internship.

Chapter 2

Internship

This chapter covers the work done during the five-month internship in the Shaman research team. Section 2.1 covers the basics related to the approach such as the general algorithm and the matching measures. Section 2.2 revolves around the work done on the prototype providing cinematographic recommendations. Section 2.3 deals with the theoretical aspects related to the generalization of the association-based approach, and the discovery of typical and atypical attributes in a set of similar items.

2.1 Detailing the Association-Based Fuzzy Approach

Following the notions presented in Section 1.4 I will start by reviewing the algorithm and the criteria used in this context, and then the matching measures.

2.1.1 Algorithm

The main idea is the following one: given a target actor, a few similarity criteria and a matching measure, it is possible to compute similar actors. While it may seem more sensible to recommend movies, actors were considered because they are more compatible with fuzzy set theory. Indeed, there are no intuitive criteria of similarity representable by fuzzy sets for movies. Actors can star only once in a movie, so there is no notion of frequency applicable to a given movie. Let us review the Algorithm 1 for this purpose.

The first part of the algorithm consists in selecting the elements typically associated with the target object c, and this must be done for every criterion i considered, n being the number of criteria selected. From these elements the multi-sets $E_i(c)$ associated with c can be computed (Line 4) and then the fuzzy sets $T_i(c)$ can be computed from $E_i(c)$ with a membership function (Line 5). Then, for every potential similar item x (Line 7), the same steps as above must be executed: computation of the multi-sets $E_i(x)$ for every criterion selected (Line 9) and then computation of the fuzzy sets $T_i(x)$ (Line 10). After that every x can be compared to c through the comparisons between the fuzzy sets $T_i(c)$ and $T_i(x)$ for every i between 1 and n (Line 11). A matching degree between the two fuzzy sets is obtained for every i, and the smallest of them all is kept as the matching degree between the two sets globally (Line 13). Finally if this degree μ is high enough – depending on the tolerance of the α selected beforehand – then the element x is deemed close enough to c to be considered as a similar item and is added to the set $S(c)$ (Line 15).

Input: a target object c; n specifications of multisets (i.e., n subqueries);
a threshold $\alpha \in (0, 1]$
Output: a fuzzy set $S(c)$ of objects similar to c

```
1  begin
2      S(c) ← ∅;
3      for i ← 1 to n do
4          compute E_i(c);
5          compute T_i(c) from E_i(c);
6      end
7      foreach item x in the relation concerned do
8          for i ← 1 to n do
9              compute E_i(x);
10             compute T_i(x) from E_i(x);
11             compute the degree of matching μ_i between T_i(c) and T_i(x);
12         end
13         μ ← min_{i=1..n} μ_i;
14         if μ ≥ α then
15             S(c) ← S(c) ∪ {μ/x}
16         end
17     end
18 end
```

Algorithm 1: Base algorithm

The computation of the matching measures between c and every element x is rather time-consuming – see Sub-section 2.2.4 for more detail on execution times – so it is capital to reduce the number of elements x as much as possible, and not consider every element in the relation concerned as stated in the Algorithm 1. This leads to the use of filters specific to every criterion i selected, to limit the number of items to consider as much as possible.

2.1.2 Specifics of the Database

The IMDb dataset was used to provide information for the prototype. However the original dataset is rather large – over 2Go of raw data – and when put into an SQL database we get 10Go worth of tables. This is including all tables obtained from the data on IMDb, so including the make-up staff. When cutting through most of the interesting data, and reorganising the tables – through normalizing the schema – we get down to 2Go of tables. Figure 2.1 presents most of the original database schema – obtained with the IMDbPY[1] script, though some attributes and tables are not included because we did not need them – and Figure 1.3 shows the normalized schema describing the dataset used by the prototype.

2.1.3 Similarity criteria

A similarity criterion chosen by the user enables the system to compute recommendations following an axis: in our case, it will be linked to the relationship between the primary and foreign keys of the

[1] http://imdbpy.sourceforge.net/

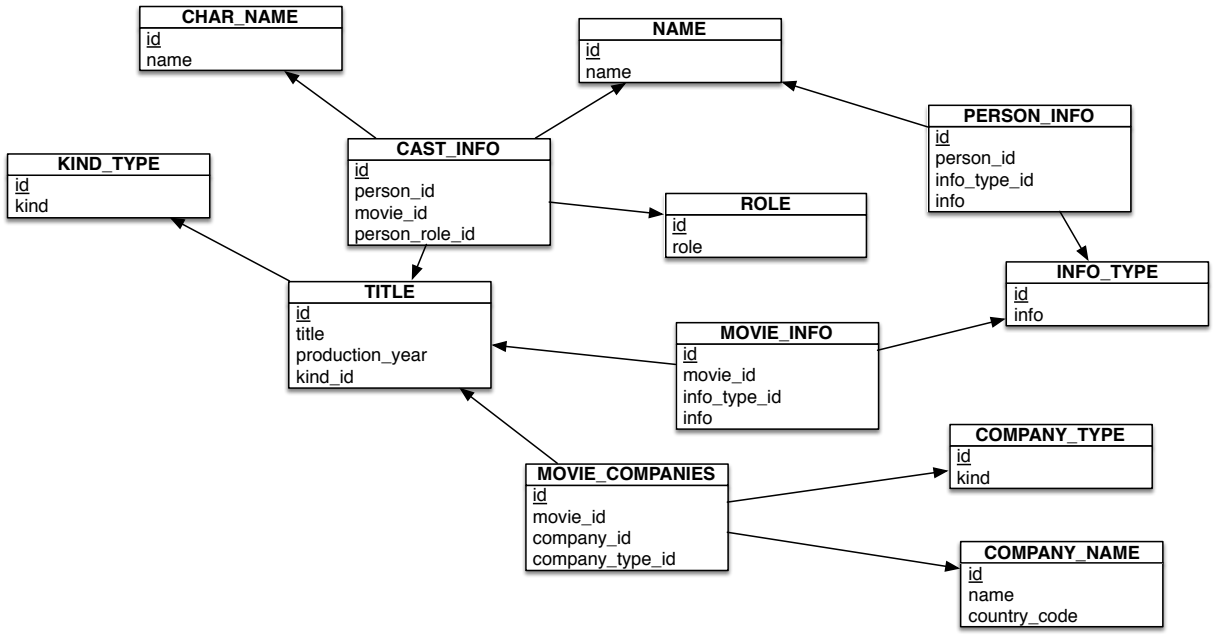

Figure 2.1: Partial schema of the original IMDb database

tables associated to this criterion. First the criterion will be used to find entities – actors, directors or genres – typically associated to the target item. Then it will help determine potential similar items – actors – with regard to said criterion, its purpose being to filter existing items. Finally the links between the selected items and the target item will be explained with the criterion. Ideally to provide better recommendations, a set of several criteria is required. While this approach is also valid with only one criterion, combining criteria is in general more interesting but it raises other challenges such as how to aggregate several matching degrees. In the case of actors, three similarity criteria have been considered: actors, directors and genres.

Actors Criterion

Two actors are considered similar if they have often played with the same other actors. In other words, if they have a set of common co-actors. Obviously, starring several times in the same movies greatly raises the likelihood to be similar with this criterion. To pass this filter the actors must have starred in more than one movie with the target actor. Thanks to this filter, the number of potential similar actors is greatly reduced compared to the number of actors in the database.

Directors Criterion

Actors may be accustomed to some directors in particular. In this case the first step is to compute the directors with whom the target actor has collaborated. Based on the number of times they have collaborated, each director will be given a frequency. Ranked according to this frequency, the most typical directors are kept for further computations. With this information, it is now possible to filter actors to retain only those who have already collaborated with at least one of the most

typical directors of the target actor. Adjusting the number of most typical directors is crucial for optimization purposes. We have set this parameter to five by default.

Genres Criterion

Actors are often attributed typical genres based on the movies they starred in. However there are not so many movie genres, and movies are often associated to several genres. In the database considered, there are more than 450.000 movies and more than 1.700.000 actors. Needless to say that this criterion is not very restrictive nor effective when used on huge databases. This is why the other criteria come first and this one last. Considering only the three most typical genres of the target actor by default, potential similar actors are given a matching degree based on the genres of the movies they played in. The potential actors are not filtered with the genres criterion as it is more a waste of time than anything else: given the low number of distinct genres and the high number of genres per movie, actors would always pass the filter anyway.

Thanks to these criteria, the number of potential similar actors is down to around a hundred for most actors. It can be less for some (around 20 for Angelina Jolie) and more for others (around 200 for Clint Eastwood). It would be interesting to adjust the default value parameters according to the number of typical values associated with the target actor, however the final number of potential similar actors cannot really be anticipated at the moment as this final number depends on the conjunction of criteria selected. As it turns out, it would appear that it is the top-3 directors that really change the top results, and the genres criterion only ever slightly changes the order. Of course weighting the criteria as explained in Sub-section 2.2.2 can change the order of the results drastically.

2.1.4 Matching Measures

Once the potential similar actors have been computed, they need to be compared one by one to the target actor. To do this, fuzzy set theory provides matching measures to compare the fuzzy sets associated with two actors. For every actor and then for every criterion, a fuzzy set of typical values is computed and then they can be compared with matching measures. There are several matching measures at our disposal, most of which have been tested before. To compare two fuzzy sets, we can either:

- test the equality of the two fuzzy sets T_1 and T_2 of (more or less) typical elements in E_1 and E_2 respectively, for example by means of the Jaccard index:

$$\mu_{matches}(E_1,\ E_2) = \frac{\sum_{x \in U} \min(\mu_{T_1}(x),\ \mu_{T_2}(x))}{\sum_{x \in U} \max(\mu_{T_1}(x),\ \mu_{T_2}(x))} \tag{2.1}$$

  where U denotes the underlying domain of E_1 and E_2, or by means of a drastic measure such as:

$$\mu_{matches}(E_1,\ E_2) = \inf_{x \in U}\ 1 - |\mu_{T_1}(x) - \mu_{T_2}(x)|. \tag{2.2}$$

- check whether there exists at least one element which is typical both in E_1 and in E_2 (which corresponds to a rather lax view):

$$\mu_{matches}(E_1,\ E_2) = \sup_{x \in U}\ \min(\mu_{T_1}(x),\ \mu_{T_2}(x)). \tag{2.3}$$

- assess the extent to which most of the elements which are typical in E_1 are also typical in E_2 and reciprocally:

$$\mu_{matches}(E_1,\ E_2) = \min(\mu_{most\ \in\ T_2}(T_1),\ \mu_{most\ \in\ T_1}(T_2)). \tag{2.4}$$

The evaluation of Formula (2.4) is based on (one of) the interpretation(s) of fuzzy quantified statements of the form $Q\ X\ A\ are\ B$ where A and B are fuzzy predicates and Q is a fuzzy quantifier. See [37, 30, 31]. The most simple interpretation was proposed by Zadeh [37] and is based on the ratio of elements which are A and B among those which are A:

$$\mu(Q\ X\ A\ are\ B) = \mu_Q\left(\frac{\sum_{x\in X} \top(\mu_A(x),\ \mu_B(x))}{\sum_{x\in X} \mu_A(x)}\right) \tag{2.5}$$

where $\top$ denotes a triangular norm, for instance the minimum. Then, Equation (2.4) rewrites (taking $\top = \min$):

$$\begin{aligned}\mu_{matches}(E_1,\ E_2) = \min(&\mu_{most}\left(\frac{\sum_{x\in X} \min(\mu_{T_1}(x),\ \mu_{T_2}(x))}{\sum_{x\in X} \mu_{T_2}(x)}\right),\\ &\mu_{most}\left(\frac{\sum_{x\in X} \min(\mu_{T_1}(x),\ \mu_{T_2}(x))}{\sum_{x\in X} \mu_{T_1}(x)}\right)).\end{aligned} \tag{2.6}$$

The lists of actors provided by the different matching measures can be compared using the Jaccard index (see formula 2.7). Considering only the actors criterion, we get the results presented in Table 2.1 for a panel of 20 actors, when considering the top-20 answers. It turns out that the measures presented in Equations 2.1 and 2.6 return the same results. Let us note that every matching measure and every criterion contribute to changing the results returned, more detail about their differences can be found in Appendix A.

$$Jaccard_{L,L'} = \frac{|L \bigcap L'|}{|L \bigcup L'|} \tag{2.7}$$

	L_1	L_2	L_3	L_4
L_1	1	0.17	0.23	1
L_2		1	0.14	0.17
L_3			1	0.23
L_4				1

Table 2.1: Comparison of the top-20 answers with the actors criterion (average computed on a panel of 20 actors).

2.2 Cinematographic Recommendations: Experiments

In this section, I will detail the explanation mechanism, a user survey and the aggregation measures. Also, execution times and a link with another recommender system are introduced.

2.2.1 Explanations

For every similarity criteria there is at least one element of explanation provided as to why an actor is returned. The first answers provided by the prototype were rather confusing as we had no idea as to what kind of links there were between the target actor and the actors returned. Implementing the explanation mechanism helped greatly with justifying the recommendations returned.

Directors Criterion

To justify how related an actor and a director are, we simply have to look for movies in which the two collaborated. Here is the query:

```
SELECT d.name, m.title FROM movies m, directors d
WHERE (d.id, m.id) IN
(SELECT md.director_id, md.movie_id FROM movies_cast mc, movies_directors md
WHERE mc.actor_id = SIMILAR_ACTOR
AND mc.movie_id = md.movie_id
AND md.director_id IN (TYPICAL_DIRECTORS))
order by d.name;
```

Ordering the results by director names and then by years is an arbitrary choice for display purposes. The link between primary and foreign keys is rather explicit in this query, with the associations between the tables *movies* and *directors* with *movies_cast* and *movies_directors*.

Actors Criterion

Actors can be related in two ways: the obvious one is that they have starred together in movies. The other one is that they have similar sets of co-starring actors. While the former helps to understand why actors are returned, the latter is used as a similarity criterion. Getting the movies in which the two actors starred is straightforward.

```
SELECT m.title, m.production_year FROM movies m
WHERE m.id IN
(SELECT mc1.movie_id FROM movies_cast mc1, movies_cast mc2
WHERE mc1.actor_id = TARGET_ACTOR
AND mc2.actor_id = SIMILAR_ACTOR
AND mc1.movie_id = mc2.movie_id)
order by m.production_year;
```

However getting the actors who played with both actors is slightly more complicated. Let's not forget that we decided that actors must have played together more than once to be considered similar. There are many joins in the query, so it can take a while to process (there are more than 1.7 million actors in the database). More detail on how much time every part takes can be found in Sub-section 2.2.4.

```
SELECT actors.id, actors.name, count(actors.id)
FROM movies_cast mc1, movies_cast mc2, movies_cast mc3, movies_cast mc4, actors
WHERE mc1.actor_id = TARGET_ACTOR
```

```
AND mc1.movie_id = mc2.movie_id
AND mc2.actor_id != TARGET_ACTOR
AND mc2.actor_id = mc3.actor_id
AND mc3.actor_id != SIMILAR_ACTOR
AND mc3.movie_id = mc4.movie_id
AND mc4.actor_id = SIMILAR_ACTOR
AND actors.id = mc2.actor_id
GROUP BY actors.id, actors.name
HAVING COUNT(DISTINCT mc2.movie_id) > 1
ORDER BY count(actors.id) DESC;
```

Genres Criterion

Figuring out the movie genres typically associated with actors is easy, and the most significant way to underline it is probably to count the number of movies of these genres in which the actor starred.

```
SELECT count(m.title) FROM movies m
WHERE m.id IN
(SELECT mg.movie_id FROM movies_genres mg, movies_cast mc
WHERE mc.actor_id = SIMILAR_ACTOR
AND mc.movie_id = mg.movie_id
AND mg.genre_id IN (TYPICAL_GENRES));
```

Other explanations were considered but to no avail due to the lack of data in the database – such as the order of importance of actors in a movie. When looking for actors similar to Tom Cruise, we can get Nicole Kidman, with the following explanation:

- Nicole Kidman played in Days of Thunder, directed by Tony Scott, director typically associated to Tom Cruise.
- Nicole Kidman and Tom Cruise played in Days of Thunder, Far and Away, Eyes Wide Shut, Stanley Kubrick: A Life in Pictures, Boffo! Tinseltown's Bombs and Blockbusters, The Queen, and August.
- Also they often played with Tom Hanks, Robert Redford, Steven Spielberg, Dustin Hoffman, and Sydney Pollack.
- Nicole Kidman played in 37 films from the genres Action, Thriller, and Drama, typically associated to Tom Cruise.

Let us note that it was fairly easy on our case to provide these explanations as it is the same mechanism behind recommendations and explanations. In [12] the authors looked for ways to explain the relationships between pairs of entities independently of the way the relationships were discovered. They used a knowledge graph based on DBpedia[2] to compute their explanations. To do this they re-used and adapted algorithms from [4, 5, 16], as the explication of paths is analog to keyword search in databases.

[2] http://wiki.dbpedia.org/About

2.2.2 Impact of the Aggregation Measure

When considering only one similarity criterion, there is no aggregation to worry about. In all other cases there is. Let us consider several similarity criteria: we will then have several matching degrees – maybe obtained with different matching measures – to aggregate, and this can be done in several ways. The original way was to take the minimum of the degrees. The answers produced can be radically different, but not with every matching measure. Table 2.2 shows results obtained with the actors considered in the preceding section, with the Jaccard index. The same matching measures are associated with L_1, L_2, L_3 and L_4, with the minimum aggregation. Concerning L_5, L_6, L_7 and L_8 the matching measures are the same as L_1, L_2, L_3 and L_4 respectively, with the arithmetic mean aggregation. It appears that the measures presented in Equations 2.2 and 2.6 return very

	L_5	L_6	L_7	L_8
L_1	0.37	0.27	0.24	0.49
L_2	0.48	0.72	0.15	0.41
L_3	0.2	0.16	0.3	0.24
L_4	0.37	0.26	0.24	0.49

Table 2.2: Impact of the aggregation on the top-20 answers (average computed on a panel of 20 actors).

similar results (0.72 and 0.49). An obvious drawback to the minimum aggregation measure is that an answer completely satisfying all criteria but one will be rejected. This is why we considered using the arithmetic mean of the degrees instead of the minimum. The aggregation does not only change the results, it can also affect the explanations generated, with changing the order of the arguments for instance and the "depth" of the justifications.

If we introduce the notion of importance for similarity criteria, we can try to better take into account the preferences of the user. The idea is no longer to simply ask the user to choose criteria, but also to rank them one way or another. We can imagine users for whom the similarity between actors is greatly induced by the fact that they starred together, quite with the directors and slightly with the genres. As a result the explanations are modified, as we show below.

It is possible to "split up" (Figure 2.2) a few sets of movies, with A_1 the target actor, A_2 the returned actor and $t5$ the five most typical directors associated to the target actor A_1 for instance. Let us insist on the fact that this principle may concern any criterion. We are going to represent these sets of movies with a tree. Let us start with the set of movies in which A_2 stars, $movies(A_2)$. If we choose to give more importance to the directors, we will be more interested in the movies in the sub-tree $movies(A_2, t5)$, corresponding to movies directed by one of the five most typical directors of A_1, in which A_2 stars. Then if we consider that it is important for the two actors to have starred together, we go down in the sub-tree $movies(A_2, t5, A_1)$. The elements that should be used to justify the recommendation would then be movies in which both A_1 and A_2 starred, directed by one of the directors of $t5$. On the contrary if the two actors starring together is more important than the director, then we will consider the tree presented in Figure 2.3.

Choosing how important is this or that criterion could be done when the user chooses his desired criteria, through ordering them for instance, or affecting them weights (very important, moderately important, slightly important, ...) with drop-down menus. Then it is when degrees are aggregated that the weights will allow us to formulate the desired compromise between the criteria. As a first implementation I chose to use a slider with predefined steps as shown in Figure 2.4.

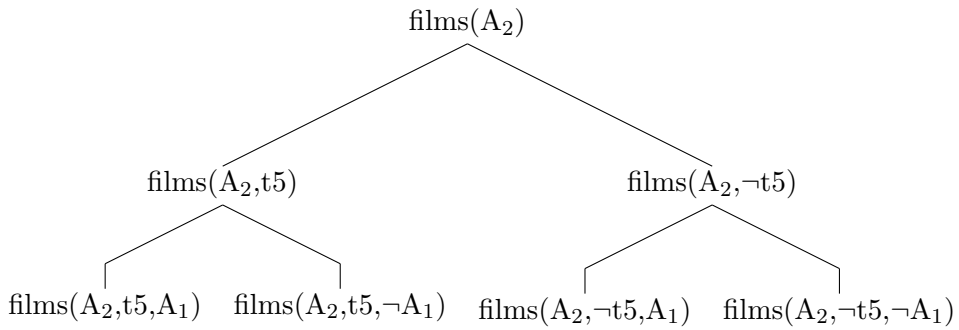

Figure 2.2: Separation according to a criterion then according to the target actor

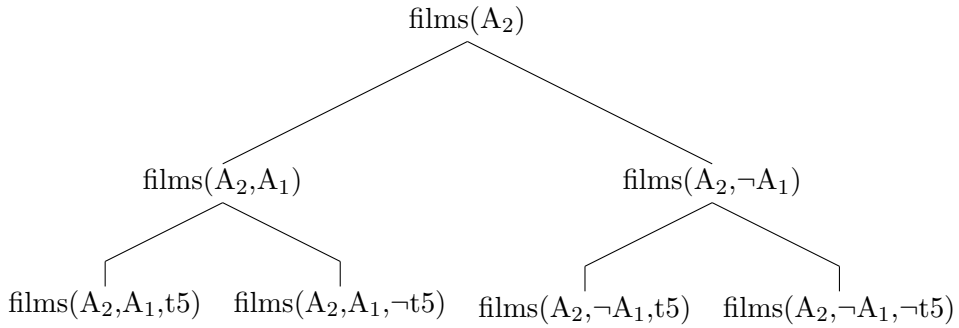

Figure 2.3: Separation according to the target actor then according to a criterion

Figure 2.4: How to customize the weights given to each criterion.

2.2.3 Experimentation

A preliminary experiment – see Sub-section 1.4.1 – was carried out in [21]. The objective was to evaluate the relevance of the recommendations provided for three given actors. Two criteria were

used – director and actor – and there were no explanations to justify the results. Apparently the Jaccard index seemed to return better results than the others with an average relevance of 58%. The rate of "I don't know" answers was fairly high, between 60% and 75% on average for every matching measure.

We conducted a new study to test the effect of the explanations on the impact of the judgment of the users (Figure 2.5). Users were asked to rate the relevance of recommended actors from a

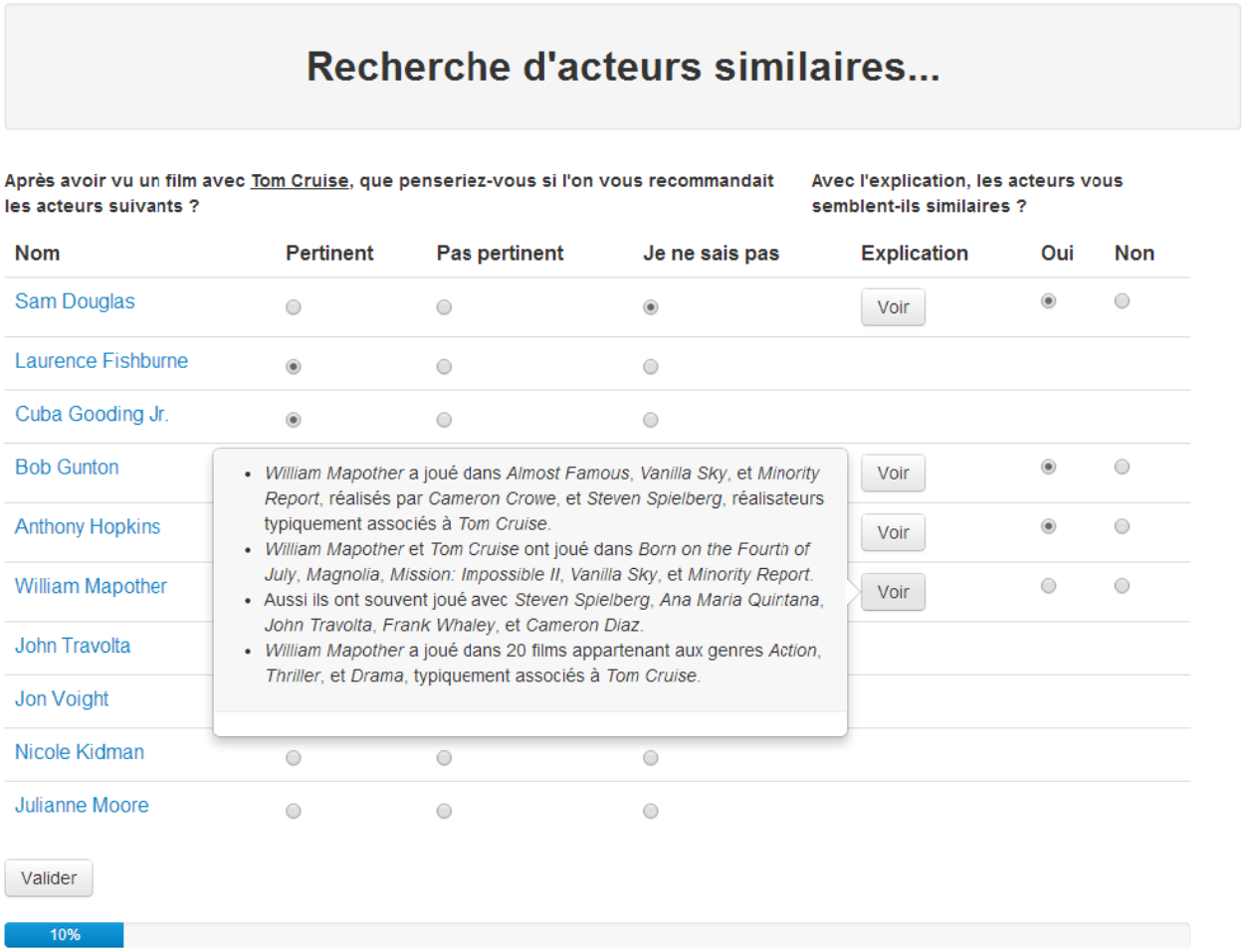

Figure 2.5: Survey screenshot.

target actor without directly being aware of the criteria used to compute the similarity. However the format of the explanation gave the user an idea of the latter. Two factors are considered here: the opinion of the user from only the names of the two actors, and the opinion of the user once he has taken into account the explanation. We considered four target actors: Tom Cruise, Johnny Depp, Hugh Jackman and Nicole Kidman. The user study is divided into four parts (one per actor). The study proposes similar actors based on the director, actor and genre criteria, with the matching measure described by equation (1.4) and aggregation by the min. The study required a couple of minutes to be completed. Twenty-one users completed the study. For all four target actors, the results presented in Table 2.3 are obtained. It shows that the proposed actors

are considered relevant on an average of 45.7 % prior to the explanation, and then 80.8 % after explanation. Figure 2.6 details the relevance scores for each actor before having explained why they were returned: the gap between Johnny Depp and Nicole Kidman is close to 25% which may be explained by their notoriety. However Figure 2.7 underlines that regardless of the actor, the average relevance is always very close to 80%, which is very comforting in the idea that the explanations are convincing.

	Before explanation	After explanation
Relevant	45.7	80.8
Not relevant	12.9	19.2
Does not know	41.4	0

Table 2.3: Evaluation of the relevance of actors, in %

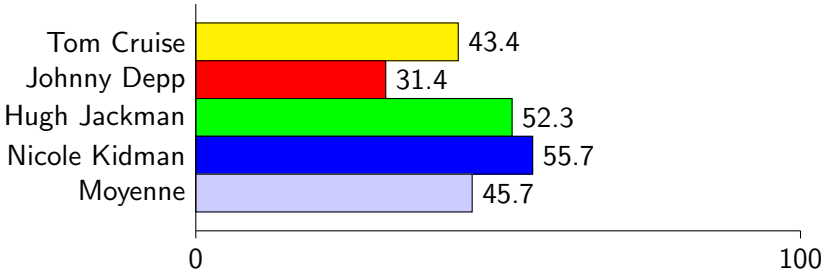

Figure 2.6: Relevance **before** explanation

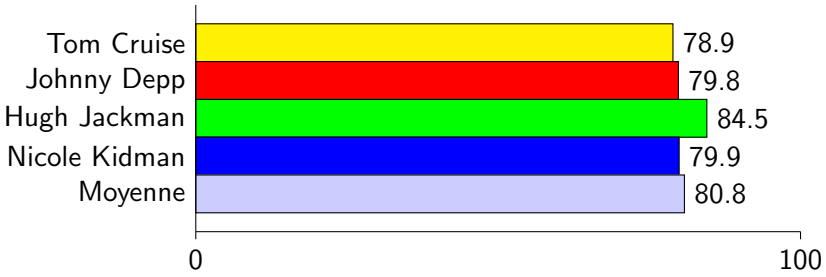

Figure 2.7: Relevance **after** explanation

2.2.4 Execution Times

Originally back in [21], considering a panel of twenty actors, the average execution time for the computation of twenty similar actors with criteria actors and directors was 1.51s, without explanations. Now the prototype is capable of giving a ranked list of recommendations and the relevant explanations (for the top-10) in less than a second on average. All three criteria are used in this case. Computing the explanations can be quite time-consuming as the more criteria are used, the

more links need to be explained. The average time spent is shown in Table 2.4. The difference is the place of the database, whether on the same computer as the prototype or on a remote server. As there are on average 360 queries to compute 10 similar actors, the network cost is more important with a remote database. However a remote server dedicated to computing data has the advantage of having specific memory management mechanisms adapted to database queries computation.

The RBDMS used here is PostgreSQL. Several mechanisms need to be taken into account when measuring execution times, including the cache, the buffers, and the opening of the connection. Configuring correctly the server is crucial to obtaining very reasonable execution times.

More specifically, the algorithm is divided into four parts: finding the target actor in the database (Name), looking for similar candidates (Criteria), computing the typical sets (Matching), and explaining the results (Expl). The average execution times can be found in Table 2.5. Considering the huge amount of time taken by the computation of the fuzzy sets to compare the elements, the emphasis is on reducing the number of potential candidates as much as possible. The computation of the explanations can be quite time-consuming as well, but the number of explanations is adjustable depending on the desired context. Furthermore, the user would probably be interested in the most similar elements only.

N°	Actor name	Nb actors compared	Remote time (ms)	Local time (ms)
1	Tom Cruise	68	1089.0	701.0
2	Julia Roberts	85	912.0	626.0
3	Robin Williams	84	1061.0	765.0
4	Brad Pitt	73	814.0	571.0
5	Jean Reno	86	685.0	456.0
6	Alain Delon	193	1581.0	1124.0
7	Thierry Lhermitte	201	1757.0	1216.0
8	Jim Carrey	51	563.0	404.0
9	Morgan Freeman	107	1086.0	818.0
10	Meryl Streep	64	803.0	639.0
11	Bruce Willis	145	1308.0	985.0
12	Cameron Diaz	47	475.0	340.0
13	Sigourney Weaver	63	662.0	500.0
14	Christian Bale	39	413.0	285.0
15	Vincent Cassel	68	511.0	344.0
16	Dustin Hoffman	115	1316.0	1039.0
17	Jodie Foster	26	431.0	349.0
18	Al Pacino	135	1073.0	789.0
19	Jack Black	103	981.0	723.0
20	Anthony Hopkins	138	1364.0	1061.0
	Average	94.55	944.25	686.75

Table 2.4: Average number of actors compared and time with a remote and local database

2.2.5 Recommendation Systems

Several kinds of recommendation systems were introduced in the state of the art, another is based on demographic data. The idea is to use data related to people to figure out elements that would

Database	Name	Criteria	Matching	Expl	Total
local	2.9	28.2	485.1	162.85	679.1
remote	6.0	38.05	689.4	201.1	934.85

Table 2.5: Average execution time of each part of the algorithm

appeal to certain categories of people. Gender, age, profession and area are attributes considered by the MovieLens[3] group providing such data on movies ratings. Their datasets can be utilized to compute collaborative filtering recommendations. Beyond this approach, it is also possible to use the association-based approach: we could be interested in movies liked by people typically associated with the movies we like. The similarity criteria taken into account here would be the age, the occupation and the area for instance. Let us consider the movie *Star Wars: Episode IV - A New Hope*. We are going to look for movies typically liked by people who liked this movie. So let us look for the users who gave the maximum rating to this movie, and then compute the multi-sets associated with the age and occupation of these users. According to the experiment, it is mostly users aged 25-34 or students who have liked this movie. It turns out that the preferred movies of this category of users include *Total Recall*, *Star Wars: Episode V - The Empire Strikes Back* and *Terminator 2: Judgment Day* for instance.

2.3 Computing the Atypical Attributes of Similar Elements

Did you know that Michael Jackson used to be pretty good at karate? While this may seem unusual and rather out of the blue, this is an example of unexpected information given by the authors of [29]. They use popularity and unpopularity to match objects so they may figure out how expected or unexpected some relationships may be. Our approach to compute similar objects also resides in knowledge discovery: it is not simply about recommending similar items, it is also about highlighting the differences between these items to present the user with cooperative answers. By defining atypicality we can figure out which properties stand out in a set of somewhat similar items, and what makes these items different.

2.3.1 Atypical Attributes in a Set of Typical Elements

Beyond simply providing similar items, we saw that explaining why they are recommended is crucial. A new step is to provide information on what distinguishes two items that are similar on a given set of similarity criteria. Google for instance proposes similar search thanks to the knowledge graph, and when looking for Tom Cruise we get a few other people recommended as shown in Figure 2.8. Two of these people – Katie Holmes and Nicole Kidman – are mentioned as "Former spouse", which differentiates them from actors linked to Tom Cruise – Brad Pitt or Emily Blunt.

Typicality has been studied in several fields including cognitive psychology (see [15, 23, 20]) and fuzzy logic ([32, 36, 13]). We consider typicality here as Zadeh did in [38], however the following work should be applicable to any interpretation of typicality. Zadeh defined x as a typical element in a fuzzy set Y iff x has a high membership degree in Y and most of the elements of Y are similar to x. When Y is a normal set, x belongs to it if most elements of Y are similar to x. While the notion of typicality is not limited to the use of frequencies, Dubois and Prade created a typicality

[3] http://grouplens.org/datasets/movielens/

measure based on frequency and similarity in [10]. On the other hand, atypicality is a concept mostly studied in cognitive psychology and less so in computer science. Most of its computations rely on attribute comparisons, and they do not seem to have been used with associations. Using in a first time the same similarity criteria used in the previous section, we will then use new associations to highlight the atypicality of certain properties associated with some items. One obvious issue resides in finding which associations should be used to look for atypical properties, and actually having enough diverse data to do this.

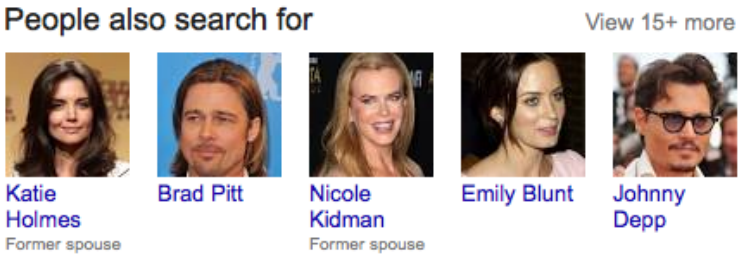

Figure 2.8: Example of results associated to Tom Cruise with Google Search

Let us consider a database D and a target object x. Let us choose a few similarity criteria – or associations – to compute elements typically associated with x. For each criterion c we will compute for x a multiset $E_c(x)$ and then a fuzzy set $T_c(x)$ representing how typical the elements are in $E_c(x)$. After having filtered the potential similar items, multisets and fuzzy sets are computed for them too, one for each similarity criterion c considered. In order to compare them, we compute and compare the fuzzy sets and we get a matching degree. With these degrees we can now rank the potential similar items and only retain the top-k. Let us call this set of similar items Y, and choose $k = 5$, we get $Y = [y_1, y_2, y_3, y_4, y_5]$. As detailed in Sub-section 2.2.1, explanations are then formulated to justify how close each similar item $y \in Y$ is to the target object x. Now we desire to highlight what distinguishes every element y from the set Y. We are going to consider the similarity criteria used before to compute the similar items. For every item y we have already computed the multisets and fuzzy sets $E_c(y)$ and $T_c(y)$ associated with each criterion c. The objective is to compute the fuzzy sets $T_c(Y)$ associated with each criterion c, representing how typical the elements are in the multiset $E_c(Y)$. Here are two ways to do this:

- compute $E_c(Y)$ from the multisets $E_c(y)$ – through a multiset sum, see equation 2.8 – and then compute $T_c(Y)$;

$$E_c(Y) = \biguplus E_c(y) \tag{2.8}$$

- compute $T_c(Y)$ directly from the $T_c(y)$ – through an arithmetic mean, see equation 2.9.

$$T_c(Y) = \{\mu/x_i, \mu = \frac{1}{|Y|} \sum_{y \in Y} \mu_y, \mu_y/x_i \in T_c(y)\} \tag{2.9}$$

Let us consider the following fuzzy sets associated with the items y:
$T_c(y_1) = \{0.1/a,\ 0.2/b,\ 0.05/e,\ 0.1/k\}$
$T_c(y_2) = \{0.1/b,\ 0.2/c,\ 0.05/h,\ 0.1/i\}$
$T_c(y_3) = \{0.1/a,\ 0.2/f,\ 0.05/g,\ 0.1/k\}$

$T_c(y_4) = \{0.1/d,\ 0.2/g,\ 0.05/h,\ 0.1/k\}$
$T_c(y_5) = \{0.1/e,\ 0.2/f,\ 0.05/g,\ 0.1/h\}$
With Equation 2.9 the resulting fuzzy set associated with the set Y is:
$T_c(Y) = \{0.04/a,\ 0.06/b,\ 0.04/c,\ 0.02/d,\ 0.03/e,\ 0.08/f,\ 0.06/g,\ 0.4/h,\ 0.02/i,\ 0.06/k\}$

Once this fuzzy set $T_c(Y)$ has been computed, we can start comparing each element y to the set Y thanks to the fuzzy sets $T_c(y)$ and $T_c(Y)$, the objective being to find differences between them to distinguish y from other similar elements. As $T_c(y)$ and $T_c(Y)$ can be assimilated to vectors, we can compute the difference between each coordinates and retain the highest differences obtained. Remark: it may be interesting to separate two categories of elements here: elements which are prominently present in y and absent in Y, and those prominently present in Y and absent in y. As actors tend to have rather dense careers, and to collaborate with many directors and fellow actors, there may be many of them with which only one of the similar actors would have interacted. Should this number of individuals be too high for a given similar actor, selecting an adequate number of atypical properties would be needed to limit the results. A contrario, should the distribution of the results be not sparse enough, thresholds indicating from which point a property becomes atypical would also be needed.

One starting point would be to look at the distribution of the values in $T_c(Y)$ and compute the arithmetic mean of the typicality degrees for instance. Then, all values above this mean should represent properties rather typical of the set. Any item y which would have a 0 associated with this property may stand out. On the other hand, values which are below the arithmetic mean should represent properties rather typical of only one or very few items. Considering the values above, we get a mean of 0.045 for the values of $T_c(Y)$. The properties b, f, g, k are all above the arithmetic mean, and f is the highest. Let us now compute the difference between the typicality degrees of $T_c(Y)$ and each $T_c(y)$. The results are presented in Table 2.6. The highest degrees are highlighted in green, and refer to properties that are far above the mean of the typicality degrees, so typical of the set $T_c(y)$. The lowest ones are highlighted in orange, and refer to properties rather typical of $T_c(Y)$.

	y1	y2	y3	y4	y5
a	0.06	-0.04	0.06	-0.04	-0.04
b	0.14	0.04	-0.06	-0.06	-0.06
c	-0.04	0.16	-0.04	-0.04	-0.04
d	-0.02	-0.02	-0.02	0.08	-0.02
e	0.02	-0.03	-0.03	-0.03	0.07
f	-0.08	-0.08	0.12	-0.08	0.12
g	-0.06	-0.06	-0.01	0.14	-0.01
h	-0.04	0.01	-0.04	0.01	0.06
i	-0.02	0.08	-0.02	-0.02	-0.02
k	0.04	-0.06	0.04	0.04	-0.06

Table 2.6: Difference degrees between each item

While the computation of the differences is possible for the aforementioned similarity criteria – used to compute the similar actors in the first place – it should be noted that new criteria could be used to compute differences as well. Indeed there may be some criteria that do not seem

interesting to be used to compute similarity, but that would provide interesting information on atypical properties. Beyond associations, it could also be attributes associated with the actor such as their date of birth. In the example presented below, it could be noted that Nicole Kidman is the only woman in the set of actors recommended. Also, other data sources would provide valuable information to highlight atypical properties. Bar charts representing the value of each property for each actor would highlight which ones are singular. Two issues detailed in the next Sub-section include data sources and choosing the interesting properties. It is also possible to compute atypical properties that define x with regard to the set of similar elements Y.

Algorithm 2 describes a method to compute these "unique" properties that distinguish objects. The first part (Lines 3 to 17) is the same as in Algorithm 1. Then for every criterion c the fuzzy set $T_c(Y)$ is computed (Line 19). For every element $y \in Y$ the differences between y and Y are computed (Line 21). Finally the differences between the target object x and the set of similar items Y are computed (Line 23).

Input: a target object x ; n specifications of multisets (i.e., n subqueries) ;
Output: a set of distinguishable properties for each object y similar to x

```
1  begin
2      Y(x) ← ∅;
3      for i ← 1 to n do
4          compute E_i(x);
5          compute T_i(x) from E_i(x);
6      end
7      foreach item y in the relation concerned do
8          for i ← 1 to n do
9              compute E_i(y);
10             compute T_i(y) from E_i(y);
11             compute the degree of matching μ_i between T_i(x) and T_i(y);
12         end
13         μ ← min_{i=1..n} μ_i;
14         if μ ≥ α then
15             Y(x) ← Y(x) ∪ {μ/y}
16         end
17     end
18     for c ← 1 to n do
19         compute T_c(Y) from E_c(Y) or the different T_c(y)
20         foreach item y in Y(x) do
21             Compute the difference between T_c(y) and T_c(Y) for each property and return the
               highest
22         end
23         Compute the biggest differences between T_c(x) and T_c(Y)
24     end
25 end
```

Algorithm 2: How to compute atypical elements

Let us consider the IMDb database, and look for actors similar to a target actor x, say Tom

Cruise. Let us choose a few similarity criteria – or associations – to compute elements typically associated with x. We can choose the actors, directors and genres criteria for instance, and for each criterion c we will compute for x a multiset $E_c(x)$ and then a fuzzy set $T_c(x)$ representing how typical the elements are in $E_c(x)$. After having filtered the potential similar actors, multisets and fuzzy sets are computed for them too, one for each similarity criterion c considered. In order to compare them, we compute and compare the fuzzy sets and we get a matching degree. With these degrees we can now rank the potential similar actors and only retain the top-k. Let us call this set of similar actors Y, and choose $k = 5$. We get $Y =$ [Nicole Kidman, Steven Spielberg, William Mapother, Tom Hanks, Tim Robbins]. Let us consider the genres criterion. For every genre g and for every actor y, a typicality degree $\alpha_{y,g}$ has been computed. They are shown in Table 2.7.

Then the difference between each typicality degree and the typicality degrees of $T_{genres}(Y)$ – with equations 2.8 and 2.9 – is computed for each actor. A positive number will indicate that the actor is typically more associated with this genre than the set of actors, while a negative number will indicate the opposite. These numbers are presented in Table 2.8 for Equation 2.8 and in Table 2.9 for Equation 2.9, and the highest and lowest values for each actor are highlighted. It turns out that some of these most significant values are different depending on the equation used to compute $T_c(Y)$. Thus, the prototype will provide indications such as:

- Unlike other actors similar to Tom Cruise, Steven Spielberg played mostly in movies of the genre Documentary.
- Unlike other actors similar to Tom Cruise, Steven Spielberg did not play much in movies of the genre Drama.
- Unlike other actors similar to Tom Cruise, Tom Hanks played mostly in movies of the genre Comedy.

It is also possible to provide information as to what makes the target actor unique compared to the recommendations provided. This can be done the very same way with computing the difference between the typicality degrees of the target actor with those of the set of similar actors. Before finding the atypical properties of items, finding the similarity criteria on any given database would be rather useful, which is the topic of the next Sub-section.

2.3.2 Self-Discovery of Similarity Criteria

Fully automatizing this whole program independently of the database may very well be impossible. The notion of similarity in itself is far too subjective to be adapted to every person potentially interested in using this system. A semi-automatic approach would be to imagine an expert defining what similarity is in a given database, and then the program would – mostly – be able to unfold itself. There are filters for every criterion, and different uses for them as well. The reason for that is that not all criteria will have the same impact on the result. The genres criterion is a perfect example of this, as it is extremely vague and difficult to use to cut down a set of actors, unlike the actors and directors criteria. The cardinality of the relations could be used to predict the use of the criterion – the genres relation contains only a few distinct tuples, unlike the actors and directors relations which contains respectively about 1.700.000 and 400.000 tuples. Still considering the IMDb database, there are many tables – producers, editors, composers, cinematographers among others – ready to be used. But how relevant are they? While perfect for associations, they may

N°	Property	Id	Tom Cruise	Nicole Kidman	Steven Spielberg	William Mapother	Tom Hanks	Tim Robbins
0	Western	1	0,0088	0,0083	0	0	0,0072	0
1	Animation	2	0	0,0083	0	0	0,0507	0
2	War	3	0,0439	0,0165	0	0,0317	0,0217	0,015
3	History	5	0,0088	0,0083	0,0196	0,0317	0,0145	0,015
4	Musical	6	0	0,0165	0	0	0	0
5	Documentary	8	0,0526	0,0661	0,3333	0,0317	0,0725	0,0677
6	Sci-Fi	9	0,0263	0,0083	0,0392	0,0476	0	0,0451
7	Short	10	0	0	0,0392	0,0476	0,0145	0
8	Thriller	11	0,114	0,124	0,0588	0,127	0,058	0,0677
9	Biography	13	0,0263	0,0331	0,0588	0,0476	0,0217	0,015
10	Horror	14	0,0088	0,0165	0,0392	0,0635	0,0145	0,0075
11	Action	15	0,114	0,0331	0,0784	0,0635	0,0145	0,0526
12	Comedy	16	0,0789	0,0744	0,098	0,0635	0,2029	0,1955
13	News	17	0	0	0	0,0159	0	0,0075
14	Family	18	0	0,0165	0	0	0,0435	0,015
15	Adventure	19	0,0877	0,0413	0,0588	0,0159	0,058	0,0677
16	Romance	21	0,0877	0,124	0,0196	0,0159	0,087	0,0902
17	Drama	22	0,2105	0,2231	0,0392	0,2063	0,1667	0,188
18	Fantasy	23	0,0088	0,0413	0,0392	0	0,0435	0,0301
19	Sport	24	0,0351	0,0083	0	0,0476	0,0145	0,0075
20	Mystery	25	0,0439	0,0496	0,0196	0,0635	0,0217	0,0301
21	Crime	26	0,0439	0,0661	0,0196	0,0635	0,0507	0,0526
22	Music	27	0	0,0165	0,0392	0,0159	0,0217	0,0301
	Max		0,2105	0,2231	0,3333	0,2063	0,2029	0,1955

Table 2.7: Typicality of each actor similar to Tom Cruise, for each genre concerned

not seem interesting at first glance, especially to provide actors recommendations. Yet they might be interesting to highlight some atypical properties of some actors. So how should these criteria, properties and associations be chosen? There are far too many possibilities for this to result in acceptable computing times.

Considering a database with knowledge of the schema and the cardinalities of the relations, let us review how it would be possible to automatize the algorithm. The first step is to identify from which table the target object will come from. Then to identify which tables are linked to this object thanks to primary and foreign keys. These tables will all be potential criteria for now, but they can still be discarded for two reasons: either the information they contain is not relevant – i.e. not interesting according to the expert – or despite a primary-foreign key relationship it cannot be used by an association-based approach – such as a table containing personal data e.g. date of birth. An intervention from the expert would be very interesting to cut down the number of tables used for criteria computation, so as to limit the number of potential criteria. Another idea would be to have a look at the paths used by the queries already formulated by users to have an idea of how users combine tables to acquire results, such as in [35, 17, 33] resulting in the creation of schema summaries. Then depending on the cardinality of the remaining tables, use as many of

N°	Property	Id	Nicole Kidman	Steven Spielberg	William Mapother	Tom Hanks	Tim Robbins	Set Y
0	Western	1	0,0043	-0,004	-0,004	0,0033	-0,004	0,004
1	Animation	2	-0,0075	-0,0158	-0,0158	0,0349	-0,0158	0,0158
2	War	3	-0,0013	-0,0178	0,014	0,004	-0,0027	0,0178
3	History	5	-0,0075	0,0038	0,0159	-0,0013	-0,0008	0,0158
4	Musical	6	0,0126	-0,004	-0,004	-0,004	-0,004	0,004
5	Documentary	8	-0,0248	0,2424	-0,0592	-0,0184	-0,0232	0,0909
6	Sci-Fi	9	-0,0155	0,0155	0,0239	-0,0237	0,0214	0,0237
7	Short	10	-0,0138	0,0254	0,0338	0,0007	-0,0138	0,0138
8	Thriller	11	0,039	-0,0262	0,042	-0,027	-0,0173	0,085
9	Biography	13	0,0034	0,0292	0,018	-0,0079	-0,0146	0,0296
10	Horror	14	-0,0052	0,0175	0,0418	-0,0072	-0,0142	0,0217
11	Action	15	-0,0084	0,0369	0,022	-0,027	0,0111	0,0415
12	Comedy	16	-0,0679	-0,0443	-0,0788	0,0606	0,0532	0,1423
13	News	17	-0,004	-0,004	0,0119	-0,004	0,0036	0,004
14	Family	18	-0,0032	-0,0198	-0,0198	0,0237	-0,0047	0,0198
15	Adventure	19	-0,0101	0,0074	-0,0355	0,0066	0,0163	0,0514
16	Romance	21	0,0429	-0,0614	-0,0652	0,0059	0,0092	0,081
17	Drama	22	0,0453	-0,1386	0,0285	-0,0112	0,0101	0,1779
18	Fantasy	23	0,0077	0,0056	-0,0336	0,0099	-0,0035	0,0336
19	Sport	24	-0,0056	-0,0138	0,0338	0,0007	-0,0063	0,0138
20	Mystery	25	0,014	-0,016	0,0279	-0,0138	-0,0055	0,0356
21	Crime	26	0,0128	-0,0338	0,0101	-0,0026	-0,0007	0,0534
22	Music	27	-0,0072	0,0155	-0,0078	-0,002	0,0064	0,0237

Table 2.8: Actor's typicality minus the set Y's typicality for each genre (computed with Equation 2.8)

them as possible as filters to cut down the number of potential candidates for similarity elements, and finally use all criteria for matching purposes.

However the greatest challenge persists in the usability of the database: if the primary-foreign key relationships are not explicit then this association-based approach may not be adaptable. As mentioned in Sub-section 2.1.2 it is possible to refine the database, but that does not sound really possible with production databases.

N°	Property	Id	Nicole Kidman	Steven Spielberg	William Mapother	Tom Hanks	Tim Robbins	Set Y
0	Western	1	0,0052	-0,0031	-0,0031	0,0041	-0,0031	0,0031
1	Animation	2	-0,0035	-0,0118	-0,0118	0,0389	-0,0118	0,0118
2	War	3	-0,0005	-0,017	0,0147	0,0047	-0,002	0,017
3	History	5	-0,0096	0,0018	0,0139	-0,0033	-0,0028	0,0178
4	Musical	6	0,0132	-0,0033	-0,0033	-0,0033	-0,0033	0,0033
5	Documentary	8	-0,0481	0,2191	-0,0825	-0,0418	-0,0466	0,1143
6	Sci-Fi	9	-0,0198	0,0112	0,0196	-0,028	0,0171	0,028
7	Short	10	-0,0203	0,019	0,0274	-0,0058	-0,0203	0,0203
8	Thriller	11	0,0369	-0,0283	0,0399	-0,0291	-0,0194	0,0871
9	Biography	13	-0,0022	0,0236	0,0124	-0,0135	-0,0202	0,0353
10	Horror	14	-0,0117	0,011	0,0352	-0,0138	-0,0207	0,0282
11	Action	15	-0,0154	0,03	0,0151	-0,0339	0,0042	0,0484
12	Comedy	16	-0,0525	-0,0288	-0,0634	0,076	0,0686	0,1269
13	News	17	-0,0047	-0,0047	0,0112	-0,0047	0,0028	0,0047
14	Family	18	0,0015	-0,015	-0,015	0,0285	0	0,015
15	Adventure	19	-0,007	0,0105	-0,0325	0,0096	0,0193	0,0483
16	Romance	21	0,0566	-0,0477	-0,0515	0,0196	0,0229	0,0673
17	Drama	22	0,0585	-0,1255	0,0417	0,002	0,0233	0,1647
18	Fantasy	23	0,0105	0,0084	-0,0308	0,0127	-0,0007	0,0308
19	Sport	24	-0,0073	-0,0156	0,032	-0,0011	-0,0081	0,0156
20	Mystery	25	0,0127	-0,0173	0,0266	-0,0152	-0,0068	0,0369
21	Crime	26	0,0156	-0,0309	0,013	0,0002	0,0021	0,0505
22	Music	27	-0,0082	0,0145	-0,0088	-0,0029	0,0054	0,0247

Table 2.9: Actor's typicality minus the set Y's typicality for each genre (computed with Equation 2.9)

Conclusion

This report presented a state of the art on databases and information retrieval, followed with the presentation of the work done during the Master's internship. The original prototype now offers cinematographic recommendations based on several criteria, that may be computed with several matching and aggregation measures, and that may be explained. A few notions on recommender systems, schema summarization and similarity measures were introduced to give a scope on the possibilities of their uses in our subject. Fusing classical recommender systems with our approach appears to be feasible, having the advantage of providing explanations as to why elements are suggested in addition to offering another way to compute elements. To improve the quality of our recommendations as cooperative answers, we looked into highlighting the atypical properties of each item recommended with regard to the whole set of similar items. So beyond simply offering recommendations as well as explaining them, we can also demonstrate how different they are from each other. This concept may be useful when facing overspecialization in content-based recommender systems or, as we introduced it, favorises knowledge discovery. Completely generalizing this method to any database automatically does not seem possible at the moment, considering the subjectivity of the notion of similarity according to users. A possibility that we will be investigating later resides in the use of domain ontologies and query logs to define which associations of tables may be interesting for the users and potentially suggest queries.

While this report heavily focused on relational databases, the underlying concepts such as similarity and cooperative answering are also applicable to other contexts, including distributed data sources. This is also why I am going to pursue this study in this new context during a Ph.D. with the same supervisors, looking for enriched answers to database queries in a distributed context. The objectives include analyzing the data sources to guarantee the quality of the answers provided, summarizing the answers returned, detect suspect answers and deal with plethoric answers, among many others!

Appendix A

Appendix

Let us recall the matching measures used. To compare two fuzzy sets, we can either:

- test the equality of the two fuzzy sets T_1 and T_2 of (more or less) typical elements in E_1 and E_2 respectively, for example by means of the Jaccard index:

$$\mu_{match}(E_1,\ E_2) = \frac{\sum_{x \in U} \min(\mu_{T_1}(x),\ \mu_{T_2}(x))}{\sum_{x \in U} \max(\mu_{T_1}(x),\ \mu_{T_2}(x))} \tag{A.1}$$

  where U denotes the underlying domain of E_1 and E_2

- check whether *every value* in U is as typical to E_1 as to E_2 (drastic version) :

$$\mu_{latch}(E_1,\ E_2) = \inf_{x \in U}\ 1 - |\mu_{T_1}(x) - \mu_{T_2}(x)|. \tag{A.2}$$

- check whether there exists at least one element which is typical both in E_1 and in E_2 (which corresponds to a rather lax view):

$$\mu_{match}(E_1,\ E_2) = \sup_{x \in U} \min(\mu_{T_1}(x),\ \mu_{T_2}(x)). \tag{A.3}$$

- assess the extent to which most of the elements which are typical in E_1 are also typical in E_2 and reciprocally:

$$\mu_{match}(E_1,\ E_2) = \min(\mu_{most\, \in\, T_2}(T_1),\ \mu_{most\, \in\, T_1}(T_2)). \tag{A.4}$$

  Equation (A.4) can be rewritten (choosing $\top = \min$) :

$$\begin{aligned} \mu_{match}(E_1,\ E_2) = \min(&\mu_{most}\left(\frac{\sum_{x \in X} \min(\mu_{T_1}(x),\ \mu_{T_2}(x))}{\sum_{x \in X} \mu_{T_2}(x)}\right), \\ &\mu_{most}\left(\frac{\sum_{x \in X} \min(\mu_{T_1}(x),\ \mu_{T_2}(x))}{\sum_{x \in X} \mu_{T_1}(x)}\right)). \end{aligned} \tag{A.5}$$

L_1 corresponds to (A.1), L_2 to (A.2), L_3 to (A.3), and L_4 to (A.5), with the minimum for aggregation measure. L_5 corresponds to (A.1), L_6 to (A.2), L_7 to (A.3), and L_8 to (A.5), with the mean for aggregation measure. A set of 20 actors was considered for these measures, for all of which the top-20 answers were considered.

	L_1	L_2	L_3	L_4
L_1	1	0.05	0.01	0.93
L_2		1	0	0.05
L_3			1	0.01
L_4				1

Table A.1: Comparison of the top-20 answers, criterion DIR

	L_1	L_2	L_3	L_4
L_1	0	0	0	0
L_2		0	0	0
L_3			0	0
L_4				0

Table A.2: Comparison of the top-20 answers, criterion GEN (unusable alone)

	L_1	L_2	L_3	L_4
L_1	1	0.17	0.23	1
L_2		1	0.14	0.17
L_3			1	0.23
L_4				1

Table A.3: Comparison of the top-20 answers, criterion ACT

	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8
L_1	1	0.23	0.29	1	0.7	0.24	0.23	0.7
L_2		1	0.15	0.23	0.24	0.81	0.14	0.24
L_3			1	0.29	0.3	0.14	0.37	0.3
L_4				1	0.7	0.24	0.23	0.7
L_5					1	0.24	0.24	1
L_6						1	0.13	0.24
L_7							1	0.24
L_8								1

Table A.4: Comparison of the top-20 answers, criteria DIR ACT

	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8
L_1	1	0.07	0.01	0.92	0.14	0.06	0.01	0.23
L_2		1	0	0.07	0.22	0.58	0	0.19
L_3			1	0.01	0	0	0.37	0
L_4				1	0.14	0.06	0.01	0.23
L_5					1	0.25	0.01	0.64
L_6						1	0	0.21
L_7							1	0.01
L_8								1

Table A.5: Comparison of the top-20 answers, criteria DIR GEN

	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8
L_1	1	0.17	0.23	1	0.24	0.17	0.18	0.34
L_2		1	0.12	0.17	0.47	0.92	0.13	0.41
L_3			1	0.23	0.12	0.12	0.36	0.14
L_4				1	0.24	0.17	0.18	0.34
L_5					1	0.47	0.15	0.77
L_6						1	0.12	0.42
L_7							1	0.15
L_8								1

Table A.6: Comparison of the top-20 answers, criteria ACT GEN

	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8
L_1	1	0.26	0.29	1	0.37	0.27	0.24	0.49
L_2		1	0.17	0.26	0.48	0.72	0.15	0.41
L_3			1	0.29	0.2	0.16	0.3	0.24
L_4				1	0.37	0.26	0.24	0.49
L_5					1	0.49	0.18	0.76
L_6						1	0.15	0.42
L_7							1	0.2
L_8								1

Table A.7: Comparison of the top-20 answers, criteria DIR ACT GEN

Table A.8 presents the typicality degrees of the actors for some of the directors associated with them. There were 191 rows originally, but for space reasons only a few were retained. The directors which had been associated with only one actor only once were removed as they are not exactly significant with regard to the set of similar actors. Unlike the typicality degrees regarding the genres presented in Table 2.7, this table is obviously more sparse considering the high number of directors associated with each actor. Tables A.9 and A.10 contain the typicality degrees of the directors minus the mean of the set for each actor, according to Equations 2.8 and 2.9.

N°	Property	Id	Tom Cruise	Nicole Kidman	Steven Spielberg	William Mapother	Tom Hanks	Tim Robbins
9	Luhrmann, Baz	590825	0	0,0351	0	0	0	0
12	De Palma, Brian	232937	0,0227	0	0	0	0,0164	0,0164
22	Ephron, Nora	1263417	0	0,0175	0	0	0,0328	0
23	Benton, Robert	79556	0	0,0351	0	0	0	0
36	Zemeckis, Robert	1087310	0	0	0	0	0,0492	0
37	Dante, Joe	221793	0	0	0,0303	0	0,0164	0
55	Robbins, Tim	827850	0	0	0	0	0	0,0328
62	Duigan, John	267408	0	0,0351	0	0	0	0
66	Spielberg, Steven	929203	0,0455	0	0,0909	0,04	0,0656	0,0164
69	Couturié, Bill	203189	0,0227	0,0175	0,0303	0	0,0164	0,0164
70	Unkrich, Lee	1002842	0	0	0	0	0,0328	0
96	Demme, Jonathan	241582	0	0	0	0	0,0164	0,0164
114	Lasseter, John	552214	0	0	0	0	0,0492	0
123	Roach, Jay	827318	0,0227	0	0,0303	0	0	0,0164
131	Darabont, Frank	222106	0	0	0	0	0,0164	0,0164
135	Noyce, Phillip	718821	0	0,0175	0	0	0	0,0164
142	Scott, Tony	884080	0,0455	0,0175	0	0	0	0,0164
145	Marshall, Penny	1462607	0	0	0	0	0,0328	0
147	Altman, Robert	20533	0	0	0	0	0	0,0492
150	Stone, Oliver	943616	0,0227	0	0	0,08	0	0
154	Howard, Ron	441619	0,0227	0,0175	0	0	0,0656	0
155	Davids, Paul	224787	0	0	0,0303	0	0,0164	0
166	Coen, Joel	186317	0	0	0	0	0,0164	0,0164
169	Frears, Stephen	324894	0,0227	0,0175	0,0303	0	0,0164	0,0164
172	Coen, Ethan	186310	0	0	0	0	0,0164	0,0164
177	Harlan, Jan	403058	0,0227	0,0175	0,0303	0	0	0
189	Crowe, Cameron	208985	0,0455	0	0,0303	0,08	0	0

Table A.8: Extract of the typicality of the directors associated with the actors

N°	Property	Nicole Kidman	Steven Spielberg	William Mapother	Tom Hanks	Tim Robbins	Mean Set
9	Luhrmann, Baz	0,0266	-0,0084	-0,0084	-0,0084	-0,0084	0,0084
12	De Palma, Brian	-0,0084	-0,0084	-0,0084	0,008	0,008	0,0084
22	Ephron, Nora	0,0049	-0,0127	-0,0127	0,0201	-0,0127	0,0127
23	Benton, Robert	0,0266	-0,0084	-0,0084	-0,0084	-0,0084	0,0084
36	Zemeckis, Robert	-0,0127	-0,0127	-0,0127	0,0365	-0,0127	0,0127
37	Dante, Joe	-0,0084	0,0219	-0,0084	0,008	-0,0084	0,0084
55	Robbins, Tim	-0,0084	-0,0084	-0,0084	-0,0084	0,0243	0,0084
62	Duigan, John	0,0266	-0,0084	-0,0084	-0,0084	-0,0084	0,0084
66	Spielberg, Steven	-0,038	0,0529	0,002	0,0276	-0,0216	0,038
69	Couturié, Bill	0,0007	0,0134	-0,0169	-0,0005	-0,0005	0,0169
70	Unkrich, Lee	-0,0084	-0,0084	-0,0084	0,0243	-0,0084	0,0084
96	Demme, Jonathan	-0,0084	-0,0084	-0,0084	0,008	0,008	0,0084
114	Lasseter, John	-0,0127	-0,0127	-0,0127	0,0365	-0,0127	0,0127
123	Roach, Jay	-0,0084	0,0219	-0,0084	-0,0084	0,008	0,0084
131	Darabont, Frank	-0,0084	-0,0084	-0,0084	0,008	0,008	0,0084
135	Noyce, Phillip	0,0091	-0,0084	-0,0084	-0,0084	0,008	0,0084
142	Scott, Tony	0,0091	-0,0084	-0,0084	-0,0084	0,008	0,0084
145	Marshall, Penny	-0,0084	-0,0084	-0,0084	0,0243	-0,0084	0,0084
147	Altman, Robert	-0,0127	-0,0127	-0,0127	-0,0127	0,0365	0,0127
150	Stone, Oliver	-0,0084	-0,0084	0,0716	-0,0084	-0,0084	0,0084
154	Howard, Ron	-0,0036	-0,0211	-0,0211	0,0445	-0,0211	0,0211
155	Davids, Paul	-0,0084	0,0219	-0,0084	0,008	-0,0084	0,0084
166	Coen, Joel	-0,0084	-0,0084	-0,0084	0,008	0,008	0,0084
169	Frears, Stephen	0,0007	0,0134	-0,0169	-0,0005	-0,0005	0,0169
172	Coen, Ethan	-0,0084	-0,0084	-0,0084	0,008	0,008	0,0084
177	Harlan, Jan	0,0091	0,0219	-0,0084	-0,0084	-0,0084	0,0084
189	Crowe, Cameron	-0,0127	0,0176	0,0673	-0,0127	-0,0127	0,0127

Table A.9: Actor's typicality minus the set Y's typicality for each director (computed with Equation 2.8)

N°	Property	Nicole Kidman	Steven Spielberg	William Mapother	Tom Hanks	Tim Robbins	Mean Set
9	Luhrmann, Baz	0,0281	-0,007	-0,007	-0,007	-0,007	0,007
12	De Palma, Brian	-0,0066	-0,0066	-0,0066	0,0098	0,0098	0,0066
22	Ephron, Nora	0,0075	-0,0101	-0,0101	0,0227	-0,0101	0,0101
23	Benton, Robert	0,0281	-0,007	-0,007	-0,007	-0,007	0,007
36	Zemeckis, Robert	-0,0098	-0,0098	-0,0098	0,0393	-0,0098	0,0098
37	Dante, Joe	-0,0093	0,021	-0,0093	0,0071	-0,0093	0,0093
55	Robbins, Tim	-0,0066	-0,0066	-0,0066	-0,0066	0,0262	0,0066
62	Duigan, John	0,0281	-0,007	-0,007	-0,007	-0,007	0,007
66	Spielberg, Steven	-0,0426	0,0483	-0,0026	0,023	-0,0262	0,0426
69	Couturié, Bill	0,0014	0,0142	-0,0161	0,0003	0,0003	0,0161
70	Unkrich, Lee	-0,0066	-0,0066	-0,0066	0,0262	-0,0066	0,0066
96	Demme, Jonathan	-0,0066	-0,0066	-0,0066	0,0098	0,0098	0,0066
114	Lasseter, John	-0,0098	-0,0098	-0,0098	0,0393	-0,0098	0,0098
123	Roach, Jay	-0,0093	0,021	-0,0093	-0,0093	0,0071	0,0093
131	Darabont, Frank	-0,0066	-0,0066	-0,0066	0,0098	0,0098	0,0066
135	Noyce, Phillip	0,0108	-0,0068	-0,0068	-0,0068	0,0096	0,0068
142	Scott, Tony	0,0108	-0,0068	-0,0068	-0,0068	0,0096	0,0068
145	Marshall, Penny	-0,0066	-0,0066	-0,0066	0,0262	-0,0066	0,0066
147	Altman, Robert	-0,0098	-0,0098	-0,0098	-0,0098	0,0393	0,0098
150	Stone, Oliver	-0,016	-0,016	0,064	-0,016	-0,016	0,016
154	Howard, Ron	0,0009	-0,0166	-0,0166	0,049	-0,0166	0,0166
155	Davids, Paul	-0,0093	0,021	-0,0093	0,0071	-0,0093	0,0093
166	Coen, Joel	-0,0066	-0,0066	-0,0066	0,0098	0,0098	0,0066
169	Frears, Stephen	0,0014	0,0142	-0,0161	0,0003	0,0003	0,0161
172	Coen, Ethan	-0,0066	-0,0066	-0,0066	0,0098	0,0098	0,0066
177	Harlan, Jan	0,008	0,0207	-0,0096	-0,0096	-0,0096	0,0096
189	Crowe, Cameron	-0,0221	0,0082	0,0579	-0,0221	-0,0221	0,0221

Table A.10: Actor's typicality minus the set Y's typicality for each director (computed with Equation 2.9)

Bibliography

[1] G. Adomavicius and A. Tuzhilin. Multidimensional recommender systems: a data warehousing approach. In *Electronic commerce*, pages 180–192. Springer, 2001.

[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. 17(6):734–749, 2005.

[3] G. Adomavicius, A. Tuzhilin, and R. Zheng. Rql: A query language for recommender systems. 2005.

[4] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *Proc. of ICDE'02*, pages 5–16, 2002.

[5] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *Proc. of ICDE'02*, pages 431–440, 2002.

[6] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. 46:109–132, 2013-07.

[7] K. Chakrabarti, V. Ganti, J. Han, and D. Xin. Ranking objects based on relationships. In *Proc. of SIGMOD'06*, pages 371–382, 2006.

[8] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Query recommendations for interactive database exploration. In *Proc. of SSDBM'09*, pages 3–18, 2009.

[9] M. Drosou and E. Pitoura. YmalDB: exploring relational databases via result-driven recommendations. 22(6):849–874, 2013-12-01.

[10] D. Dubois and H. Prade. On data summarization with fuzzy sets. In *Proc. of IFSA'93*, pages 465–468, 1993.

[11] S. Fakhraee and F. Fotouhi. TupleRecommender: A recommender system for relational databases. In *2011 22nd International Workshop on Database and Expert Systems Applications (DEXA)*, pages 549–553, 2011.

[12] L. Fang, A. D. Sarma, C. Yu, and P. Bohannon. Rex: Explaining relationships between entity pairs. 5(3):241–252, 2011.

[13] M. Friedman, M. Ming, and A. Kandel. On the theory of typicality. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 03(02):127–142, 1995.

[14] T. Gaasterland, P. Godfrey, and J. Minker. An overview of cooperative answering. *J. Intell. Inf. Syst.*, 1(2):123–157, 1992.

[15] J. A. Hampton. Overextension of conjunctive concepts: Evidence for a unitary model of concept typicality and class inclusion. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 14(1):12, 1988.

[16] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *Proc. of VLDB'02*, pages 670–681, 2002.

[17] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 13–24. ACM, 2007.

[18] G. Koutrika, B. Bercovitz, and H. Garcia-Molina. Flexrecs: expressing and combining flexible recommendations. In *Proc. of SIGMOD'09*, pages 745–758, 2009.

[19] G. Koutrika, A. Simitsis, and Y. E. Ioannidis. Explaining structured queries in natural language. In F. Li, M. M. Moro, S. Ghandeharizadeh, J. R. Haritsa, G. Weikum, M. J. Carey, F. Casati, E. Y. Chang, I. Manolescu, S. Mehrotra, U. Dayal, and V. J. Tsotras, editors, *ICDE*, pages 333–344. IEEE, 2010.

[20] D. Osherson and E. E. Smith. On typicality and vagueness. *Cognition*, 64(2):189 – 206, 1997.

[21] O. Pivert, G. Smits, and H. Jaudoin. Finding similar objects in relational databases - an association-based fuzzy approach. In H. L. Larsen, M. J. Martín-Bautista, M. A. Vila, T. Andreasen, and H. Christiansen, editors, *FQAS*, volume 8132 of *Lecture Notes in Computer Science*, pages 425–436. Springer, 2013.

[22] A. Rajaraman and J. D. Ullman. *Mining of massive datasets*. Cambridge University Press, Cambridge, 2012.

[23] E. Rosch and C. B. Mervis. Family resemblances: Studies in the internal structure of categories. *Cognitive Psychology*, 7(4):573 – 605, 1975.

[24] A. Simitsis, G. Koutrika, Y. Alexandrakis, and Y. E. Ioannidis. Synthesizing structured text from logical database subsets. In A. Kemper, P. Valduriez, N. Mouaddib, J. Teubner, M. Bouzeghoub, V. Markl, L. Amsaleg, and I. Manolescu, editors, *EDBT*, volume 261 of *ACM International Conference Proceeding Series*, pages 428–439. ACM, 2008.

[25] K. Stefanidis, M. Drosou, and E. Pitoura. "You may also like" results in relational databases. In *Proc. of PersDB'09*, 2009.

[26] E. Stoica, M. A. Hearst, and M. Richardson. Automating creation of hierarchical faceted metadata structures. In C. L. Sidner, T. Schultz, M. Stone, and C. Zhai, editors, *HLT-NAACL*, pages 244–251. The Association for Computational Linguistics, 2007.

[27] W. Su, J. Wang, Q. Huang, and F. Lochovsky. Query result ranking over e-commerce web databases. *Proc. of CIKM'06*, 2006.

[28] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *PVLDB*, 4(11):992–1003, 2011.

[29] K. Tsukuda, H. Ohshima, M. Yamamoto, H. Iwasaki, and K. Tanaka. Discovering unexpected information on the basis of popularity/unpopularity analysis of coordinate objects and their relationships. pages 878–885. ACM, 2013.

[30] R. Yager. General multiple-objective decision functions and linguistically quantified statements. *International Journal of Man-Machine Studies*, 21(5):389–400, 1984.

[31] R. Yager. Interpreting linguistically quantified propositions. *International Journal of Intelligent Systems*, 9(6):541–569, 1994.

[32] R. R. Yager. A note on a fuzzy measure of typicality. *International Journal of Intelligent Systems*, 12(3):233–249, 1997.

[33] X. Yang, C. M. Procopiuc, and D. Srivastava. Summarizing relational databases. 2(1):634–645, 2009.

[34] K.-P. Yee, K. Swearingen, K. Li, and M. A. Hearst. Faceted metadata for image search and browsing. In G. Cockton and P. Korhonen, editors, *CHI*, pages 401–408. ACM, 2003.

[35] C. Yu and H. V. Jagadish. Schema summarization. In *Proceedings of the 32nd international conference on Very large data bases*, pages 319–330. VLDB Endowment, 2006.

[36] L. Zadeh. A note on prototype theory and fuzzy sets. *Cognition*, 12(3):291 – 297, 1982.

[37] L. Zadeh. A computational approach to fuzzy quantifiers in natural languages. *Computing and Mathematics with Applications*, 9:149–183, 1983.

[38] L. Zadeh. A computational theory of dispositions. *International Journal of Intelligent Systems*, 2:39–63, 1987.