Chapter 6

Understanding the Semantics of Keyword Queries on Relational Data Without Accessing the Instance.

Sonia Bergamaschi, Elton Domnori, Francesco Guerra, Silvia Rota, Raquel Trillo Lado, and Yannis Velegrakis

Abstract The simplicity of keyword queries has made them particularly attractive to the technically unskilled user base, tending to become the de-facto standard for querying on the web. Unfortunatelly, alongside its simplicity, came also the loose semantics. Researchers have for a long time studied ways to understand the keyword query semantics and retrieve the most relevant data artifacts. For the web, these artifacts were documents, thus, any semantics discovering effort was based mainly on statistics about the appearance of the keywords in the documents. Recently, there has been an increasing interest in publishing structural data on the web, allowing users to exploit valuable resources that have so far been kept private within companies and organizations. These sources support only structural queries. If they are to become available on the web and be queried, the queries will be in the form of keywords and they will have to be translated into structured queries in order to be executed. Existing works have exploited the instance data in order to build offline an index that is used at query time to drive the translation. This idea is not always possible to implement since the owner of the data source is typically not willing to allow unrestricted access to the data, or to offer resources for the index construction.

Sonia Bergamaschi
DII-UNIMORE, via Vignolese 905 Modena, IT e-mail: sonia.bergamaschi@unimore.it

Elton Domnori
DII-UNIMORE, via Vignolese 905, Modena, Italy e-mail: elton.domnori@unimore.it

Francesco Guerra
DEA-UNIMORE, v.le Berengario 51, Modena, Italy e-mail: francesco.guerra@unimore.it

Silvia Rota
DII-UNIMORE, via Vignolese 905, Modena, Italy e-mail: silvia.rota@unimore.it

Raquel Trillo
Informática e Ing. Sistemas, c/ María de Luna 50018 Zaragoza, SP e-mail: raqueltl@unizar.es

Yannis Velegrakis
University of Trento, Trento, Italy e-mail: velgias@disi.unitn.eu

This chapter elaborates on methods of discovering the semantics of keyword queries without requiring access to the instance data. It describes methods that exploit meta information about the source data and the query in order to find semantic matches between the keywords and the database structures. These matches form the basis for translating the keyword query into a structure query.

6.1 Introduction

The birth of the Web has brought an exponential growth to the amount of the information that is freely available to the Internet population, overloading users and entangling their efforts to satisfy their information needs. Web search engines such as Google, Yahoo! or Bing have become popular mainly due to the fact that they offer an easy to use query interface (that is based on keywords) and an effective and efficient query execution mechanism.

The majority of these search engines do not consider information stored on the *Deep* or *Hidden Web* [9, 28], despite the fact that the size of the Deep Web is estimated to be much bigger than the *S*urface Web [9, 47]. There has been a number of systems that record interactions with the deep web sources or automatically submit queries to them (mainly through their web form interfaces) in order to index their context. Unfortunately, this technique is only partially indexing the data instance. Moreover it is not possible to take advantage of the query capabilities of data-sources, e.g., of the relational query features, because their interface is often restricted from the web-form. Besides, web search engines focus on retrieving documents and not on querying structured sources, so they are unable to access information based on concepts [31].

There are, on the other hand, numerous companies and organizations that have lots of data, so far in their own private (structured) data sources, and are willing to make the public on the web. These data will remain unexploited unless they can be queried through keyword-based interfaces which is the de-facto standard on the web. There are already various studies on how to offer such a service [18, 41, 44, 48, 50], but they all require prior access to the instance in order to build an index that will allow them at run time to understand what part of the database that a keyword in the query is referring to. This is not always feasible since companies and organizations may have their own restrictions and concerns about allowing an external service getting full access to the data.

A recent initiative that has attracted considerable attention is the notion of the *Semantic Web*[1]. The idea of the semantic web is to extend the data of the current web with more structured data (typically expressed in RDF) that will allow it to be used by machines as well as by people. The success of the Semantic Web depends on the existence of such kind of information, and on the ability to query it. The Semantic Web community has so far focused on ways to model this information.

[1] http://www.w3.org/standards/semanticweb/

Allowing relational databases to be published on the Semantic Web will provide the community with large amounts of data. However, it will also require the design and development of techniques and tools for querying that information in a way similar to the way it is currently done on the web, i.e., through keyword queries. This means that any effort toward supporting keyword queries for structural data is of major important for the specific community.

This chapter deals exactly with the problem of answering a keyword query over a relational database. To do so, one needs to understand the meaning of the keywords in the query, "guess" its possible semantics, and materialize them as SQL queries that can be executed directly on the relational database. The focus of the chapter is on techniques that do not require any prior access to the instance data, making them suitable for sources behind wrappers or web interfaces, or in general for sources that disallow prior access to their data in order to construct an index. The chapter describes two techniques that use semantic information and metadata from the sources, alongside the query itself, in order to achieve that. Apart from understanding the semantics of the keywords themselves, the techniques are also exploiting the order and the proximity of the keywords in the query to make a more educated guess. The first approach is based on an extension of the Hungarian algorithm [11] for identifying the data structures having the maximum likelihood to contain the user keywords. In the second approach, the problem of associating keywords into data structures of the relational source is modeled by means of a Hidden Markov Model, and the Viterbi algorithm is exploited for computing the mappings. Both techniques have been implemented in two systems called KEYMANTIC [5, 6] and KEYRY [7], respectively.

The chapter is structured as follows. First, a motivating example is presented to introduce the reader to the problem and the challenges. Then the problem statement is formally formulated (Section 6.3). Sections 6.4 and 6.5 describe the two approaches in details. Section 6.6 provides an overview of the related works and how they differ from the two approaches presented in this chapter.

6.2 Motivating Example

Let us assume that a virtual tourism district composed of a set of companies (travel agencies, hotels, local public administrations, tourism promotion agencies) wants to publish an integrated view of their tourism data about a location (see Figure 6.1). Keymantic allows users to query that data source with a two step process: firstly the keyword query is analyzed for discovering its intended meaning, then a ranked set of SQL queries, expressing the discovered meaning according to the database structure, is formulated.

Each keyword represents some piece of information that has been modeled in the database, but, depending on the design requirements of the data source, this piece might have been modeled as data or metadata. Thus, the first task is to discover what each keyword models in the specific data source and to which metadata / data may be

Person

Name	Phone	City	Email
Saah	4631234	London	saah@aaa.bb
Sevi	6987654	Auckland	eevi@bbb.cc
Edihb	1937842	Santiago	edibh@ccc.dd

Reserved

Person	Hotel	Date
Saah	x123	6/10/2009
Sevi	cs34	4/3/2009
Edihb	cs34	7/6/2009

Hotel

id	Name	Address	Service	City
x123	Galaxy	25 Blicker	restaurant	Shanghai
cs34	Krystal	15 Tribeca	parking	Cancun
ee67	Hilton	5 West Ocean	air cond.	Long Beach

City

Name	Country	Description
Shanghai	China	...
Cancun	Mexico	...
Long Beach	USA	...
New York	USA	...

Booked

Person	Rest	Date
Saah	Rx1	5/4/2009
Sevi	Rx1	9/3/2009

Restaurant

id	Name	Address	Specialty	City
Rx1	Best Lobster	25, Margaritas	Seafood	Cancun
Rt1	Naples	200 Park Av.	Italian	New York

Fig. 6.1 A fraction of a database schema with its data.

associated to. The association between keywords and database needs to be approximate: the synonymous and polysemous terms might allow the discovery of multiple intended meanings, each one with a rank expressing its relevance. Let us consider, for example, a query consisting of the keywords "Restaurant Naples". For instance, a possible meaning of the query might be "find information about the restaurant called Naples". In this case, the former keyword should be mapped into a metadata (the table Restaurant and the other one into a value of the attribute Name of the same table Restaurant. A user might have in mind other meanings for the same keywords, for example, "find the restaurants that are located in the Naples Avenue", or "in the Naples city", or "that cook Naples specialties". All these intended meanings give rise to different associations of the keyword `Naples`; attributes Address, City or Specialty of the table Restaurant. This example shows also that keywords in a query are not independent: we expect that the keywords express different features of what the user is looking for. For this reason we expect that in our example "Naples" is a value referring to an element of the Restaurant table. If the user had formulated the keyword query "Restaurant name Naples", the number of possible intended meanings would have been reduced, since the keywords `name` forces the mappings of Naples into the attribute Name of the table Restaurant. Notice that different intended meanings may generate a mapping from the same keyword both into metadata and into data values. For example, in our database restaurant is the name of a table, but it is also one of the possible values of the attribute Service in the table Hotel. Finally, the order of the keywords in a query is also another element to be taken into account since related elements are usually close. If a user asks for "Person London restaurant New York" one possible meaning of the query is that the user is looking for the restaurant in New York visited by people from London. Other permutations of the keywords in the query may generate other possible interpretations.

The second step in answering a keyword query concerns the formulation of an SQL query expressing one of the discovered intended meanings. In a database, semantic relationships between values are modeled either through the inclusion of different attributes under the same table or through join paths across different tables. Different join paths can lead to different interpretations. Consider, for instance, the keyword query "Person USA". One logical mapping is to have the word `Person` corresponding to the table Person and the word `USA` to a value of the attribute Country of the table City. Even when this mapping is decided, there are different interpretations of the keywords based on the different join paths that exist between the tables Person and City. For instance, one can notice that a person and a city are related through a join path that goes through the City attribute referring to the attribute Name in the table City (determining which people in the database are from USA), through another path that is based on the table Hotel (determining which people reserved rooms of Hotels in USA), and also through another path that is based on the table Restaurant (determining which people are reserved a table in an American restaurant).

Finding the different semantic interpretations of a keyword query is a combinatorial problem which can be solved by an exhaustive enumeration of the different mappings to database structures and values. The large number of different interpretations can be brought down by using internal and external knowledge that helps in eliminating mappings that are not likely to lead to meanings intended by the user. For instance, if one of the provided keywords in a query is `''320-463-1463''`, it is very unlikely that this keyword refers to an attribute or table name. It most probably represents a value, and in particular, due to its format, a phone number. Similarly, the keyword `''Bistro''` in a query does not correspond to a table or an attribute in the specific database. Some auxiliary information, such as a thesaurus, can provide the information that the word "bistro" is typically used to represent a restaurant, thus, the keyword can be associated to the Restaurant table.

6.3 Problem statement

Definition 6.1. A database D is a collection V_t of relational tables $R_1,R_2,\ldots,R_n$. Each table R is a collection of attributes $A_1,A_2,\ldots,A_{m_R}$, and each attribute A has a domain, denoted as $dom(A)$. Let $V_a=\{A \mid A\in R \wedge R\in V_t\}$ represent the set of all the attributes of all the tables in the database and $V_d=\{d \mid d=dom(A) \wedge A\in V_a\}$ represents the set of all their respective domains. The *database vocabulary* of D, denoted as V_D, is the set $V_D=V_t\cup V_a\cup V_d$. Each element of the set V_D is referred to as a *database term*.

We distinguish two subsets of the database vocabulary: the *schema vocabulary* $V_{SC} = V_t \cup V_a$ and the *domain vocabulary* $V_{DO} = V_d$ that concerns the instance information. We also assume that a keyword query KQ is an ordered l-tuple of keywords $(k_1, k_2, \ldots, k_l)$.

Definition 6.2. A *configuration* $f_c(KQ)$ of a keyword query KQ on a database D is an injective function from the keywords in KQ to database terms in V_D. In other words, a configuration is a mapping that describes each keyword in the original query in terms of database terms.

The reason we consider a configuration to be an injective function is because we assume that: (i) each keyword cannot have more than one meaning in the same configuration, i.e., it is mapped into only one database term; (ii) two keywords cannot be mapped to the same database term in a configuration since overspecified queries are only a small fraction of the queries that are typically met in practice [22]; and (iii) every keyword is relevant to the database content, i.e., keywords always have a correspondent database term. Furthermore, while modelling the keyword-to-database term mappings, we also assume that every keyword denotes an element of interest to the user, i.e., there are no stop-words or unjustified keywords in a query. In this paper we do not address query cleaning issues. We assume that the keyword queries have already been pre-processed using well-known cleansing techniques.

Answering a keyword query over a database D means finding the SQL queries that describe its possible semantics in terms of the database vocabulary. Each such SQL query is referred to as an *interpretation* of the keyword query in database terms. An interpretation is based on a configuration and includes in its clauses all the database terms that are part of the image[2] of the query keywords through the configuration. In the current work, we consider only select-project-join (SPJ) interpretations that are typically the queries of interest in similar works [2, 19], but interpretations involving aggregations [42] are part of our future work.

Definition 6.3. An *interpretation* of a keyword query $KQ = (k_1, k_2, \dots, k_l)$ on a database D using a configuration $f_c^*(KQ)$ is an SQL query in the form
select $A_1, A_2, \dots, A_o$ from R_1 JOIN R_2 JOIN … JOIN R_p where $A'_1{=}v_1$ AND $A'_2{=}v_2$ AND … AND $A'_q{=}v_q$
such that the following holds:

- $\forall A \in \{A_1, A_2, \dots, A_o\}$: $\exists k \in KQ$ such that $f_c^*(k){=}A$
- $\forall R \in \{R_1, R_2, \dots, R_p\}$: (i) $\exists k \in KQ$: $f_c^*(k){=}R$ or (ii) $\exists k_i, k_j \in KQ$: $f_c^*(k_i){=}R_i \wedge f_c^*(k_j){=}R_j$ $\wedge$ exists a join path from R_i to R_j that involves R
- $\forall$ "$A'{=}v$"$\in \{A'_1{=}v_1, A'_2{=}v_2, \dots, A'_o{=}v_o\}$: $\exists k \in KQ$ such that $f_c^*(k){=}dom(A') \wedge k = v$
- $\forall k \in KQ$: $f_c^*(k) \in \{A_1, A_2, \dots, A_o, R_1, R_2, \dots, R_p, dom(A'_1), \dots, dom(A'_q)\}$

The existence of a database term in an interpretation is justified either by belonging to the image of the respective configuration, or by participating in a join path connecting two database terms that belong to the image of the configuration. Note that even with this restriction, due to the multiple join paths in a database D, it is still possible to have multiple interpretations of a keyword query KQ given a certain configuration $f_c^*(KQ)$. We use the notation $\mathcal{I}(KQ, f_c^*(KQ), D)$ to refer to the set of these interpretations, and $\mathcal{I}(KQ, D)$ for the union of all these sets for a query KQ.

Since each keyword in a query can be mapped into a table name, an attribute name or an attribute domain, there are $2\Sigma_{i=1}^{n}|R_i| + n$ different mappings for each keyword, with $|R_i|$ denoting the *arity* of the relation R_i and n the number of tables in the database. Based on this, and on the fact that no two keywords can be mapped to the same database term, for a query containing l keywords, there are $\frac{|V_D|!}{(|V_D|-l)!}$ possible configurations. Of course, not all the interpretations generated by these configurations are equally *meaningful*. Some are more likely to represent the

[2] Since a configuration is a function, we use the term image to refer to its output.

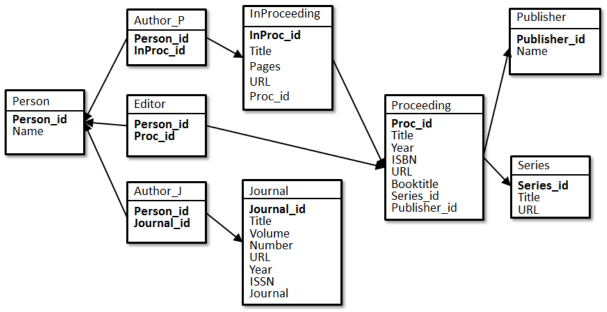

Fig. 6.2 Overview of the keyword query translation process

	R_1	...	R_n	$A_1^{R_1}$	...	$A_{n_1}^{R_1}$	...	$A_{n_n}^{R_n}$	$\underline{A_1^{R_1}}$	...	$\underline{A_{n_1}^{R_1}}$	...	$\underline{A_{n_n}^{R_n}}$
keyword$_1$	░	░	░	░	░	░	░	░	▓	▓	▓	▓	▓
keyword$_2$	░	░	░	░	░	░	░	░	▓	▓	▓	▓	▓
...	░	░	░	░	░	░	░	░	▓	▓	▓	▓	▓
keyword$_k$	░	░	░	░	░	░	░	░	▓	▓	▓	▓	▓

Fig. 6.3 Weight table with its SW (light) and VW (dark) parts

intended keyword query semantics. In the following sections we will show how different kinds of meta-information and inter-dependencies between the mappings of keywords into database terms can be exploited in order to effectively and efficiently identify these meaningful interpretations and rank them higher.

6.4 The Hungarian Algorithm Approach

The generation of interpretations that most likely describe the intended semantics of a keyword query is based on semantically meaningful configurations, i.e. sets of mappings between each keyword and a database term. We introduce the notion of *weight* that offers a quantitative measure of the relativeness of a keyword to a database term, i.e., the likelihood that the semantics of the database term are the intended semantics of the keyword in the query. The sum of the weights of the keyword-database term pairs can form a score serving as a quantitative measure of the likelihood of the configuration to lead to an interpretation that accurately describes the intended keyword query semantics. The range and full semantics of the score cannot be fully specified in advance. They depend on the method used to compute the similarity. This is not a problem as long as the same method is used to compute the scores for all the keywords. This is the same approach followed in schema matching [37] where a score is used to measure the likelihood that an element of a schema corresponds to an element of another.

	P	R	P.Na	P.Ph	P.Ci	P.Em	R.Id	R.Na	R.Ad	R.Sp	R.Ci	$\underline{\text{P.Na}}$	$\underline{\text{P.Ph}}$	$\underline{\text{P.Ci}}$	$\underline{\text{P.Em}}$	$\underline{\text{R.Id}}$	$\underline{\text{R.Na}}$	$\underline{\text{R.Ad}}$	$\underline{\text{R.Sp}}$	$\underline{\text{R.Ci}}$
people	75	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1	1	1
restaurant	0	100	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1	1	1
Naples	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1	1	1

Fig. 6.4 Intrinsic Weight SW (light gray) and VW (dark gray) matrix.

The naive approach for selecting the best configurations, and, as a consequence, generating the most prominent interpretations of a keyword query, is the computation of the score of each possible configuration and then selecting those that have the highest scores. Of course, we would like to avoid an exhaustive enumeration of all the possible configurations, and compute only those that give high scores. The problem of computing the mapping with the maximum score without an exhaustive computation of the scores of all the possible mappings is known in the literature as the problem of *Bipartite Weighted Assignments* [13]. Unfortunately, solutions to this problem suffer from two main limitations. First, apart from the mutual exclusiveness, they do not consider any other interdependencies that may exist between the mappings. Second, they typically provide only the best mapping, instead of a ranked list based on the scores.

To cope with the first limitation, we introduce two different kinds of weights: the *intrinsic*, and the *contextual* weights. Given a mapping of a keyword to a database term, its intrinsic weight measures the likelihood that the semantics of the keyword is that of the database term if considered in isolation from the mappings of all the other keywords in the query. The computation of an intrinsic weight is bed on syntactic, semantic and structural factors such attribute and relation names, or other auxiliary external sources, such vocabularies, ontologies, domains, common syntactic patterns, etc. On the other hand, a contextual weight is used to measure the same likelihood but considering the mappings of the remaining query keywords. This is motivated by the fact that the assignment of a keyword to a database term may increase or decrease the likelihood that another keyword corresponds to a certain database term. This is again bed on observations that humans tend to write queries in which related keywords are close to each other [22]. A similar idea has already been exploited in the context of schema matching [30] with many interesting results. To cope with the second limitation, we have developed a novel algorithm for computing the best mappings. The algorithm is based on and extends the Hungarian (a.k.a., Munkres) algorithm [11] and will be described in detail in Section 6.4.3.

A visual illustration of the individual steps in the keyword query translation task is depicted in Figure 6.2. A special data structure, called *weight matrix*, plays a central role in these steps. The *weight matrix* is a two-dimensional array with a row for each keyword in the keyword query, and a column for each database term. The value of a cell $[i, j]$ represents the weight associated to the mapping between the keyword i and the database term j. Figure 6.3 provides an abstract illustration of a weight matrix. An R_i and $A_j^{R_i}$ columns correspond to the relation R_i and the attribute A_j of R_i, respectively, while a column with an underline attribute name $\underline{A_j^{R_i}}$ represents the data values in the column A_j of table R_i may have, i.e., its domain. Two parts (i.e., sub-matrices) can be distinguished in the weight matrix. One corresponds to the database terms related to schema elements, i.e., relational tables and attributes, and the other one corresponds to attribute values, i.e., the domains of the attributes. We refer to database terms related to schema elements *schema databe terms*, and to those related to domains of the attributes *value database terms*. In Figure 6.3, these two sub-matrices are illustrated with different shades of gray. We refer to the weights in the first sub-matrix *schema weights*, and to those of the second *value weights*. We also use the notation SW and VW to refer either to the respective sub-matrix, or to their values. The details of the individual steps of Figure 6.2 are provided next.

Intrinsic Weight Computation. The first step of the process is the intrinsic weight computation. The output is the populated SW and VW sub-matrices. The computation is achieved by the exploitation and combination of a number of similarity techniques based on structural and lexical knowledge extracted from the data source, and on external knowledge, such ontologies, vocabularies, domain terminologies, etc. Note that the knowledge extracted from the data source is basically the meta-information that the source makes public, typically, the schema structure and constraints. In the absence of any other external information, a simple string comparison based on tree-edit distance can be used for populating the SW sub-matrix. For the VW sub-matrix the notion of *Semantic Distance* [15] can always be used in the absence of anything else. As it happens in similar situations [37], measuring the success of such a task is not easy since there is no single correct answer. In general, the more meta-information has been used, the better. However, even in the case that the current step is skipped, the process can continue with the weight matrix where all the intrinsic values have the same default value. The computation of the intrinsic weights is detailed in Section 6.4.1.

Selection of the Best Mappings to Schema Terms. The intrinsic weights provide a first indication of the similarities of the keywords to database terms. To generate the prominent mappings, we need on top of that to take into consideration the inter-dependencies between the mappings of the different keywords. We consider first the prominent mappings of keywords to schema terms. For that we work on the SW sub-matrix. Based on the intrinsic weights, a series of mappings M_1^S,M_2^S, $\ldots$, M_n^S, of keywords to schema terms are generated. The mappings are those that achieve the highest overall score, i.e., the sum of the weights of the individual keyword mappings. The mappings are partial, i.e., not all the keywords are mapped to some schema term. Those that remain unmapped will play the role of an actual data value and will be considered in a subsequent step for mapping to value database terms. The selection of the keywords to remain unmapped is bed on the weight matrix and some cut-off threshold. Those with a similarity below the threshold remain unmapped. For each of the mappings M_i^S, the weights of its SW matrix are adjusted to take into consideration the context generated by the mapping of the neighboring keywords. It is based on the observation that users form queries in which keywords referring to the same or related concepts are adjacent [22, 45]. The generation of the mappings and the adjustment of the weights in SW are performed by our extension of the Hungarian algorithm that is described in detail in Section 6.4.3. The output of such a step is an updated weight matrix SW_i and, naturally, an updated score for each mapping M_i^S. Given the updated scores, some mappings may be rejected. The selection is based on a threshold. There is no golden value to set the threshold value. It depends on the expectations from the keyword query answering systems. The higher its value, the less the interpretations generated at the end, but with higher confidence. In contrast, the lower the threshold value, the more the mappings with lower confidence.

Contextualization of VW and selection of the Best Mappings to Value Terms. For each partial mapping M_i^S of keyword to schema terms generated in the previous step, the mappings of the remaining unmapped keywords to value terms needs to be decided. This is done in two phases. First, the intrinsic weights of the VW sub-matrix that were generated in Step 1 are updated to reflect the added value provided by the mappings in M_i^S of some of the keywords to schema database terms. This is called the process of contextualization of the VW sub-matrix. It is based on the documented observation that users form queries in which keywords specifying metadata information about a concept are adjacent or at let neighboring [22, 45]. Thus, when a keyword is mapped to a schema term, it becomes more likely that an adjacent keyword should be mapped to a value in the domain of that schema term. The contextualization process increase the weights of the respective values terms to reflect exactly that. For example, in the keyword query `''Name Alexandria''` assume that the keyword `Alexandria` was found during the first step to be equally likely the name of a person or of a city. If in Step 2 the keyword `Name` has been mapped to the attribute Name of the table Person, the confidence that `Alexandria` is actually the name of a person is increased, thus, the weight between that keyword and the value database term representing the domain of attribute Name should be increased, accordingly. In the second phase, given an updated VW_i sub-matrix, the most prominent mappings of the remaining unmapped keywords to value database terms are generated. The mappings are generated by using again the adapted technique of the Hungarian algorithm (ref. Section 6.4.3). The result is a series of partial mappings M_{ik}^V, with k=1..m_i, where i identifies the mapping M_i^S on which the computation of the updated matrix VW_i w bed. Given one such mapping M_{ik}^V the value weights are further updated to reflect the mappings of the adjacent keywords to value database terms, in a way similar to the one done in Step 2 for the SW sub-matrix. The outcome modifies the total score of each mapping M_{ik}^V, and based on that score the mappings are ranked.

Generation of the Configurations. As a fourth step, each pair of a mapping M_{ik}^V together with its associated mapping M_i^S is a total mapping of the keywords to database terms, forming a configuration C_{ik}. The score of the configuration is the sum of the scores of the two mappings, or alternatively the sum of the weights in the weight matrix of the elements $[i, j]$ where i is a keyword and j is the database term to which it is mapped through M^V_{ik} or M^S_i.

Generation of the Interpretations. Having computed the best configurations, the interpretations of the keyword query, i.e., the SQL queries, can be generated. The score of each such query is the score of the respective configuration. Recall, however, that a configuration is simply a mapping of the keywords to database terms. The presence of different join paths among these terms results in multiple interpretations. Different strategies can be used to further rank the selections. One popular option is the length of the join path [21] but other heuristics found in the literature [19] can also be used. It is also possible that a same interpretation be obtained with different configurations. A post-processing analysis and the application of data-fusion techniques [10] can be used to deal with this issue. However, this is not the main focus of the current work and we will not elaborate further on it. We adopt a greedy approach that computes a query for every alternative join path. In particular, we construct a graph in which each node corresponds to a database term. An edge connects two terms if they are structurally related, i.e., through a table-attribute-domain value relationship, or semantically, i.e., through a referential integrity constraint. Given a configuration we mark all terms that are part of the range of the configuration "marked". Then we run a breath-first traversal (that favors shorter paths) to find paths that connect the disconnected components of the graph (if possible). The final SQL query is then constructed using the "marked" database terms, and in particular, the tables for its from clause, the conditions modeled by the edges for its where clause and the remaining attributes for its select clause. Then the process is repeated to find a different interpretation, that will be based on a different join path. The final order of the generated interpretations is determined by the way the different paths are discovered and the cost of the configuration on which each interpretation is based.

It is important to note here that if the thresholds used in the above steps are all brought down to zero, then our technique is able to generate all the possible interpretations that can be defined on a databe, even the most unlikely. In that sense, our technique is complete. The thresholds serve only to exclude from the results any interpretation that is not likely to represent the semantics of the keyword query, while the weights are used to provide the basis for a ranking metric.

6.4.1 Intrinsic Weight Computation

To compute the intrinsic weights, we need to compute the relevance between every query keyword and every database term. Some fundamental information that can be used towards this directions, and that is typically available, is the schema information. It may include the table and attribute names, the domains of the attributes, and very often referential and integrity constraints, such as keys and foreign keys. Syntactic descriptions of the contents of an attribute (e.g., regular expressions) can also

Algorithm 12: Intrinsic SW Matrix Computation

```
Data: $Q$: Keyword Query, $T$: Schema Database Terms
Result: SW matrix
begin
    $SW \leftarrow [0,0,\ldots,0]$ ;
    $\Sigma \leftarrow$ { Synonyms(w,t), Hyponyms(w,t), Hypernyms(w,t), StringSimilarity(w,t) ... } ;
    for $w \in Q$ do
        for $e \in T$ do
            $sim \leftarrow 0$ ;
            for $m \in \Sigma$ do
                if $m(w,e) > sim$ then
                    $sim \leftarrow m(w,e)$ ;
            if $ssim \leq threshold$ then
                $sim \leftarrow 0$;
        $SW[w,c] = ssim * 100$ ;
```

lead to a better matching of keywords to database terms since they offer indications on whether a keyword can serve as a value for an attribute or not. There are already many works that offer typical syntax for common attributes such as phone numbers, addresses, etc. [37], and have been used extensively and successfully in other areas. If access to the catalog tables is possible, assertion statements can offer an alternative source of syntactic information. In the same spirit, relevant values [8], i.e., clusters of the attribute domains, are also valuable auxiliary information. Furthermore, there is today a large volume of grammatical and semantic information that is publicly available on the Internet and can be used as a service. Examples include the popular WordNet and the many community specific ontologies.

6.4.1.1 Weights for Schema Database Terms

Finding matches between the flat list of keywords and the schema terms looks like the situation of schema matching [37] in which one of the schemas is the flat universal relation [29]. We follow a similar approach in which we employ a number of different similarity measurement techniques and consider the one that offers the best result. One of these techniques is the string similarity [16]. For the string similarity we further employ a number of different similarity metrics such as the Jaccard, the Hamming, the Levenshtein, etc., in order to cover a broad spectrum of situations that may occur. Since string similarity may fail in cases of highly heterogeneous schemas that lack a common vocabulary, we also measure the relativeness of a keyword to schema database term based on their semantic relationship. For that we employ public ontologies, such as SUMO[3], or semantic dictionaries such as Word-

[3] www.ontologyportal.org

Net, that can provide synonyms, hypernyms, hyponyms, or other terms related to a given word.

Algorithm 12 describes the computation procedure of the intrinsic schema weight matrix SW. The set Σ represents the similarity methods we employ. We have a number of default methods that represent the state of the art in the area, but additional methods can be included. Each such method takes as input two strings and returns their respective similarity in a range between 0 and 1. We trust the method that gives the highest similarity. If a similarity between a keyword and a schema term is found below a specific threshold (that is set by the application) then the similarity is set explicitly to 0. As a result, at the end of the procedure there might be rows in the matrix SW containing only zeros. These rows represent keywords that are not similar enough to any of the schema database terms, thus, they will be considered later as candidates for mapping to value database terms, i.e., domains of the schema attributes. The fact that their rows in the SW matrix are 0 instead of some low value, makes the similarities of the keyword to the value database terms that will be computed in a later step to be the dominating factor determining the guess on the role a specific keyword can play.

Example 6.1. Consider the keyword query "people restaurant Naples" posed on the database of Figure 6.1. Figure 6.4 illustrates a fraction of the weight matrix containing the intrinsic weights for the database terms derived from the tables Person and Restaurant. Instead of the full names of tables and attributes, only the first letter of the tables and the first two letters of the attributes are used. The schema weights SW are the light gray colored part of the matrix. Note that the keyword `Naples` has not been mapped to any of the schema terms since all the values of its row in SW are 0.

6.4.1.2 Weights for Value Database Terms

For computing the intrinsic value weights, we mainly exploit domain information, and base our decision on whether a keyword belongs to the domain of an attribute or not. Furthermore, we have adapted the notion of *Semantic Distance* [15] that is based on results retrieved by a search engine in order to evaluate the relatedness of two concepts. In particular, we define the *semantic relatedness* $SR(x,y)$ of two terms x and y as:

$SR(x,y){=}e^{-2NXD(x,y)}$, where

$NXD(x,y){=}\{max\{logf(x),logf(y)\}-logf(x,y)\}/\{logN-min\{logf(x),logf(y)\}\}$

with $f(x)$ denoting the number of web documents containing x, and $f(x,y)$ the number of documents containing both x and y, as these numbers are reported by specific search engines such as Google, Yahoo!, Cuil, Excite!, etc. The number N represents the number of documents indexed by the corresponding search engine. For our purpose, we compute the semantic relatedness of every keyword - attribute domain pair and this gives us an indication of the similarity degree between the keyword and the attribute domain. Information about possible values that an attribute can accept is also an important factor. The information is based on the explicit enumeration of values, as in the *Relevant Values* approach [8]. When a keyword is found among (or is similar to) the valid values that an attribute can get, the keyword receives a high weight. Additional comparison techniques include semantic measures based on external knowledge bases.

Example 6.2. For the keyword query introduced in Example 6.1, the intrinsic value weights are indicated in the VW part of Figure 6.4 i.e., the dark gray-colored part. These weights have been computed by using domain knowledge and regular expressions. Note that these are the value weights, thus, the similarity is not between the keyword and the name of the attribute, but concerns the compatibility of the keyword with the attribute domain.

6.4.2 Contextualization

The process of contextualization, as previously explained, exploits the interdependencies across mappings of different keywords. There are three different forms of contextualization that we consider. The first one increases the confidence of a keyword corresponding to an attribute (respectively, relation), if an adjacent keyword is mapped to the relation it belongs (respectively, one of the attributes of the relation). This is based on the generally observed behavior that users may sometimes provide more than one specification for a concept in a keyword query. For instance, they may use the keyword `Person` before the keyword `Name` to specify that they refer to the name of a person. The second form of contextualization is similar to the first, but applies on the value weights instead of the schema weights. The third and most important contextualization form is the one that updates the confidence of certain keywords corresponding to value database terms based on the mappings of other keywords to schema terms. The process consists of three phases and takes as input the value weight matrix VW and a partial mapping of keywords to schema database terms, and returns an updated matrix VW. Let K be the ordered list of keywords in a query, M_i^S a partial mapping of keywords in K to schema terms, and K^S the subset of K containing the keywords for which M_i^S is defined, i.e., those that are mapped to some schema terms. We define the notion of *free trailing keywords* of a keyword k, denoted as $T(k)$, to be the maximum set k_1, k_2, $\dots k_m$, of consecutive keywords in K that are between two keywords k_s and k_e in the query and for which k_s=k, M_i^S is defined for k_s and k_e and undefined for every k_i with i=1..m. The notion of *free preceding keywords* of a keyword k, denoted as $P(k)$, is defined accordingly, with k playing the role of k_e.

As an initialization step, all the weights in the rows of VW corresponding to keywords already mapped to database terms are set to zero. This is done to guarantee that in the three phases that are described next, none of these keywords will be mapped to a value term. In the first phase of the contextualization, for every keyword k mapped to a relation R through M_i^S, the weights of the trailing and preceding keywords $T(k)$ and $P(k)$ for terms representing the domains of the attributes of the relation R are increased by a constant value Δw. The rational of this action is that queries typically contain keywords that are generic descriptions for the values they provide. For instance, "`Person Bill`", "`Restaurant Naples`", etc., which means that consecutive keywords may correspond to a relation and a value of one of that relation's attributes. During the second phase, for every keyword k mapped to an attribute A through M_i^S, the weights of the trailing and preceding keywords $T(k)$ and $P(k)$ with the database terms representing domains of attributes in the same relation as A are also increased by a constant value Δw. The rational of this rule is that consecutive keywords may represent value specifications and related values. An example is the query "`Restaurant Vesuvio Pizza`" intended to ask about the restaurant "Vesuvio" that has the "Pizza" as specialty. In the third phase, if a keyword k is mapped to an attribute A, the weights of the trailing and preceding keywords related to domains of attributes related to A through some join path are increased by the constant value Δw. The rational is that users use keywords referring to concepts that are semantically related, even if these concepts have been modeled in the database in different tables. An example of this situation is the keyword query "`Phone Tribeca`". If `phone` is mapped to the attribute Phone of the relation Person, then the keyword `Tribeca` most likely represents the address of the department, which is stored in a separate table, and then the keyword query is about finding the phone number of the department that is on Tribeca. Note that the weight increase Δw can also be a percentage, instead of a constant, but our experimentations have showed no significant differences.

6.4.3 Selecting the Best Mappings

Given a weight matrix, computing the best possible mapping of keywords to database terms is known as the *assignment problem* [13]. Unfortunately, the traditional solutions return the first best mapping while we would like them all in descending order, or at least the top-k. Furthermore, we need a solution that, during the computation of the best mappings, takes into consideration inter-dependencies of the different assignments, i.e., the contextualization process. For this reason, we have adapted the popular systematic Hungarian, a.k.a. Munkres, algorithm [11] in order not to stop after the generation of the best mapping but to continue to the generation of the second best, the third, etc. Furthermore, some of its internal steps have been modified so that the weight matrix is dynamically updated every time that a mapping of a keyword to a database term is decided during the computation.

The execution of the algorithm consists of a series of iterative steps that generate a mapping with a maximum score. Once done, the weight matrix is modified accordingly to exclude the mapping that was generated and the process continues to compute the mapping with the second largest score, etc. More specifically, the maximum weight of each row is first identified and characterized as *maximum*. If the characterized as maximum weights are all located in different columns, then a mapping is generated by associating for each of the characterized as maximum weights the keyword and the database term that correspond to its respective row and column. On the other hand, if there is a column containing more than one weight characterized as maximum, all maximums in the column except the one with the maximum value loose their characterization as maximum. This last action means that some of the rows are left without some characterized weight. The values of the weights in these rows are then updated according to a number of contextual rules mentioned in Section 6.4. This is the effect of the mapped keywords to those that have remained unmapped.

In the sequel, for each row with no characterized weight, the one with the maximum value that belongs to a column corresponding to a database term different from the previous one is selected and characterized as *maximum*. If this leads to a matrix that has each characterized weight in a different column, then a mapping is generated as above or the process of un-characterizing some of the values as previously described is repeated.

Once a mapping of all the keywords has been formed, it is included in the set of mappings that will be returned by the algorithm. Then, the algorithm needs to be repeated to generate additional mappings. To avoid recomputing the same assignments, the algorithm is re-executed cyclically with a new matrix as input. The new matrix is the old one modified to exclude mappings that have already been considered in previous runs of the algorithm. This is done by setting the values of the respective weights to a large negative number, forcing the algorithm to never select them again. This whole process is repeated until the scores of the mappings that the algorithm generates fall below some specific threshold. By construction, the most prominent mapping is the one that is first reported by this task.

The original Hungarian algorithm for rectangular matrices has an $O(n^2 * m)$ complexity [11], where n is the number of keywords and m is the number of databases terms. Extending the algorithm to consider dynamic weights as described above brings the complexity to $O(n^3 * m^2)$ which is due to the fact that a mapping may affect any other mapping, thus, in the worst case, $(n-1)*(m-1)$ weight updates may take place. Nevertheless, this worst case rarely happens since only a subset of the matrix is updated in each iteration, and, due to the threshold, not all the possible updates are evaluated.

Algorithm 13 depicts the overall process of computing the set of most prominent mappings of a set of keywords to database terms, given a weight matrix. The expression $HUNGARIAN_{ext}$ refers to our extended version of the Hungarian algorithm.

Example 6.3. Figure 6.5 illustrates a VW matrix similar to the one of Figure 6.4, but with additional keywords to better demonstrate the steps described in this section. The initial version of the matrix is the one composed of the first 5 lines. The lines of the keywords people and restaurant will remain unchanged since the keywords are mapped to schema terms, and for this reason they are omitted from the subsequent versions. The weights in white cells are those characterized as maximum. Each one is the largest weight in its row. Note that for column $\underline{\text{R.Sp}}$ there are more than one

Algorithm 13: Keyword to db term mapping selection

```
Data: $I(i_{ij})$ where $I \equiv SW$ or $I \equiv VW$
Result: $M^I = \{M^I_1, \ldots, M^I_z\}$: Mappings generated by $I$

MAPPING($I$, $W_{MAX}$);
begin
    $tempM = \bigcup i_{pt} \leftarrow HUNGARIAN_{Ext} * (I)$;
    $W \leftarrow \sum i_{pt}$ ;
    $M^I \leftarrow tempM$;
    if ($W > c * W_{MAX}$) then
        $W_{MAX} \leftarrow W$;
    while ($W > c * W_{MAX}$) do
        for $i_{pt} \in tempM$ do
            $i_{pt} \in I \leftarrow -100$;
            $Mapping(I, W_{MAX})$;
```

weight characterized as maximum. From those, only the largest one is kept, in our case 55. This leaves the row of keyword `Naples` with no weight characterized as maximum. The result is the second VW matrix illustrated in Figure 6.5. The three characterized weights suggest a mapping for the keywords `Vesuvio`, and `pizza`. Given these mappings, the weights are adjusted to reflect the inter-dependencies according to the contextual rules. For instance, the mapping of `Vesuvio` to the database term $\underline{\text{R.Na}}$, triggers an increase in the weights of the database terms on the attributes in the same table. The result of firing the contextual rules is the third matrix in Figure 6.5. In the updated matrix, the highest value is 49, which is in the column $\underline{\text{R.Sp}}$, but cannot be chosen since the keyword `pizza` is already mapped to it. The second largest weight, 45, is in a column of a database term that no keyword is mapped to, and it is becoming characterized as maximum. The final outcome is the fourth matrix of Figure 6.5 and, based on this, the mappings of keywords `Vesuvio`,`pizza`, and `Naples` to the database terms $\underline{\text{R.Na}}$, $\underline{\text{R.Sp}}$, and $\underline{\text{R.Ci}}$, respectively, which is added to the mapping results generated by the algorithm. After that, the mapping is again considered for generating a new input for the algorithm. Four new matrices are derived from it, each one having one of the weights that was in a white cell in the last matrix reduced. For each obtained matrix, the whole process starts again from the beginning.

6.5 The Viterbi Approach

In an effort to define the configuration function we can divide the problem of matching a whole query to database terms into smaller sub-tasks. In each sub-task the best match between a single keyword and a database term is found. Then the final solution to the global problem is the union of the matches found in the sub-tasks. This

	P.Na	P.Ph	P.Ci	P.Em	R.Id	R.Na	R.Ad	R.Sp	R.Ci
people	0	0	0	0	0	0	0	0	0
restaurant	0	0	0	0	0	0	0	0	0
Vesuvio	50	0	50	0	0	75	50	50	50
pizza	35	0	10	0	0	32	40	55	45
Naples	30	0	40	0	0	20	25	50	45

Vesuvio	50	0	50	0	0	75	50	50	50
pizza	35	0	10	0	0	32	40	55	45
Naples	30	0	40	0	0	20	25	50	45

Vesuvio	45	0	45	0	0	75	55	55	55
pizza	30	0	5	0	0	37	45	55	50
Naples	30	0	40	0	0	20	25	55	50

Vesuvio	50	0	50	0	0	75	50	50	50
pizza	35	0	10	0	0	32	40	55	45
Naples	30	0	40	0	0	20	25	50	45

Fig. 6.5 Weight matrix during best mapping computation

approach works well when the keywords in a query are independent of each other, meaning that they do not influence the match of the other keywords to database terms. Unfortunately, this assumption does not hold in real cases. On the contrary, inter-dependencies among keywords meanings are of fundamental importance in disambiguating the keyword semantics.

In order to take into account these inter-dependencies, we model the matching function as a sequential process where the order is determined by the keyword ordering in the query. In each step of the process a single keyword is matched against a database term, taking into account the result of the previous keyword match in the sequence. This process has a finite number of steps, equal to the query length, and is stochastic since the matching between a keyword and a database term is not deterministic: the same keyword can have different meanings in different queries and hence being matched with different database terms; vice-versa, different database terms may match the same keyword in different queries. This type of process can be modeled, effectively, by using a Hidden Markov Model (HMM, for short), that is a stochastic finite state machine where the states are hidden variables.

A HMM models a stochastic process that is not observable directly (it is hidden), but it can be observed indirectly through the observable symbols produced by another stochastic process. The model is composed of a finite number N of states. Assuming a time-discrete model, at each time step a new state is entered based on a *transition probability distribution*, and an observation is produced according to an *emission probability distribution* that depends on the current state, where both these distributions are time-independent. Moreover, the process starts from an initial state based on an *initial state probability distribution*. We will consider first order HMMs with discrete observations. In these models the *Markov property* is respected, i.e., the transition probability distribution of the states at time $t+1$ depends only on the current state at time t and it does not depend on the past states at time $1,2,\dots,t-1$. Moreover, the observations are discrete: there exists a finite number,

M, of observable symbols, hence the emission probability distributions can be effectively represented using *multinomial distributions* dependent on the states. More formally, the model consists of: (i) a set os states $S = \{s_i\}$, $1 \leq i \leq N$; (ii) a set of observation symbols $V = \{v_j\}$, $1 \leq j \leq M$; (iii) a transition probability distribution $A = \{a_{ij}\}$, $1 \leq i \leq N$, $1 \leq j \leq N$ where

$$a_{ij} = P(q_{t+1} = s_j | q_t = s_i) \text{ and } \sum_{0<j<N} a_{ij} = 1$$

(iv) an emission probability distribution $B = \{b_i(m)\}$, $1 \leq i \leq N$, $1 \leq m \leq M$ where

$$b_i(m) = P(o_t = v_m | q_t = s_i) \text{ and } \sum_{0<m<M} b_i(m) = 1$$

and (v) an initial state probability distribution $\Pi = \{\pi_i\}$, $1 \leq i \leq N$ where

$$\pi_i = P(q_1 = s_i) \text{ and } \sum_{0<i<N} \pi_i = 1$$

Based on the above, the notation $\lambda = (A, B, \Pi)$ will be used to indicate a HMM. In our context, the keywords inserted by the user are the observable part of the process, while the correspondent database terms are the unknown variables that have to be inferred. For this reason, we model the keywords as observations and each term in the database vocabulary as a state.

6.5.1 Setting HMM parameters

In order to define a HMM, its parameters have to be identified. This is usually done using a training algorithm that, after many iterations, converges to a good solution for the parameter values. In our approach we introduce some heuristic rules that allow the definition of the parameter values even when no training data is available. The HMM parameter values are set by exploiting the semantics collected from the data source metadata. In particular:

The transition probabilities are computed using heuristic rules that take into account the semantic relationships that exist between the database terms (aggregation, generalization and inclusion relationships). The goal of the rules is to foster the transition between database terms belonging to the same table and belonging to tables connected through foreign keys. The transition probability values decrease with the distance of the states, e.g. transitions between terms in the same table have higher probability than transitions between terms in tables directly connected through foreign keys, that, in turn, have higher probability than transitions between terms in tables connected through a third table.

The emission probabilities are computed on the basis of similarity measures. In particular two different techniques are adopted for the database terms in V_{SC} and

V_{DO}. We use the well known *edit distance* for computing the lexical similarity between the keywords and each term (and its synonyms extracted from Wordnet[4]) in the schema vocabulary V_{SC}. On the other side, the similarity between keywords and the terms in the domain vocabulary V_{DO} is based on domain compatibilities and regular expressions. We use the calculated similarity as an estimate for the conditional probability $P(q_t = s_i | o_t = v_m)$ then, using the Bayes theorem, we calculate the emission probability $P(o_t = v_m | q_t = s_i)$. Note that the model is independent of the similarity measure adopted. Other more complex measures that take into account external knowledge sources (i.e., public ontologies and thesauri) can be applied without modifying the model.

The initial state probabilities are estimated by means of the scores provided by the HITS algorithm [25]. The HITS algorithm is a link analysis algorithm that calculates two different scores for each page: *authority* and *hub*. A high authority score indicates that the page contains valuable information with respect to the user query, while a high hub score suggests that the page contains useful links toward authoritative pages. This algorithm has been adapted to our context in order to rank the tables in a database based on their authority scores. The higher the rank, the more valuable is the information stored in the tables. For this reason, the authority score is used as an estimate of the initial state probabilities.

6.5.1.1 Adapted HITS algorithm

In order to employ the HITS algorithm, we build the database graph $G_D = (V_t, E_{fk})$ which is a directed graph where the nodes are the database tables in V_t and the edges are connections between tables through foreign keys, i.e., given two tables R_i, R_j there exists an edge in E_{fk} from R_i to R_j, denoted as $R_i \to R_j$, if an attribe in R_j is referred by a foreign key defined in R_i. Let A be the n x n adjacency matrix of the graph G_D

$$A = [a_{ij}],\ a_{ij} = 1 \text{ iff } e_{ij} \in E_{fk},\ a_{ij} = 0 \text{ iff } e_{ij} \notin E_{fk}$$

In our approach we use the modified matrix B that takes into account the number of attributes minus the number of foreign keys in a table (foreign keys are considered as links).

$$B = [b_{ij}],\ b_{ij} = a_{ij} \cdot (|R_i| - \|\{R_i \to R_j, 1 \le j \le n\}\|)$$

Let us define the authority weight vector *auth* and the hub weight vector *hub*

$$auth^T = [u_1, u_2, \ldots, u_n] \ \text{ and } \ hub^T = [v_1, v_2, \ldots, v_n]$$

The algorithm initializes these vectors with uniform values, e.g., $\frac{1}{n}$, then the vectors are updated in successive iterations as follows

$$\begin{cases} auth = B^T \cdot hub \\ hub = B \cdot auth \end{cases}$$

[4] http://wordnet.princeton.edu

At each iteration a normalization step is performed to obtain authority and hub scores in the range $[0,1]$. After few iterations the algorithm converges and the authority scores are used as estimate for the initial state probabilities.

Example 6.4. Let us consider the database in Figure 6.1. This database generates 52 states, one for each database term. Concerning the transition probability distribution, the heuristic rules foster transitions between database terms of the same table or in tables connect via foreign key. According to this, the most probable states subsequent to the state associated to the table *People* are the states associated to the names and the domains of the attributes *Name*, *Phone*, *City*, etc., then, the state associated to the tables *Booked*, *Reserved* and *City*, subsequently, the states associated to the tables *Hotel* and *Restaurant*. We use different similarity measures for computing the emission probability distribution. Domain compatibility and regular expressions are used for measuring the probabilities of the values associated to states of terms in the V_{DO}. The probabilities of values associated to states if terms in the V_{SC} is computed on the basis of lexical similarity computed on the term and on its synonyms, hypernyms and hyponyms. For example, we consider "Brasserie" as synonym of restaurant.

6.5.2 Decoding the HMM

By applying the Viterbi algorithm to an HMM, we find the state sequence which has the highest probability of generating the observation sequence. If we consider the user keyword query as an observation sequence, the algorithm retrieves the states (i.e. the data source elements) that more likely represent the intended meaning of the user query. Note that, in general, the N best sequences, each one terminating in a different state, found by the Viterbi algorithm during the recursion are not the top-N best sequences. The Viterbi algorithm divides the solution space (the number of configurations between l keywords and d database terms is $P(l,d) = \frac{l!}{(l-d)!}$) into N sub-spaces, where each sub-space contains all the sequences that end in state S_i, $1 \leq i \leq N$. The algorithm finds the best solution in each sub-space, then orders them to find the best overall solution. This process guarantees to find only the single best solution.

The Viterbi algorithm is not a good choice for facing the general problem of finding the top-K solutions. The algorithm finds all K solutions only in the particular case where $K \leq N$ and each sub-space contains 0 or 1 of the top-K solutions. In the worst scenario, when all the top-K solutions are concentrated in a single sub-space, the algorithm finds just the single best one. Moreover, we do not know in advance how the K solutions are distributed in the N sub-spaces. In order to solve the general problem of finding the overall best K sequences, we implemented a generalization of the Viterbi algorithm, called f, that keeps track of the best K solutions in each sub-space and guarantees to find the top-K sequences even in the worst scenario.

Our implementation of List Viterbi is presented in pseudo-code in Algorithm 14. The algorithm is a generalization of the Viterbi algorithm because for $K = 1$ it finds the same solution as the original algorithm, and for $K = \frac{l!}{(l-d)!}$ it finds all the existing solutions and orders them globally. The algorithm is based on a tree structure that memorizes the sequences found (each sequence starts from the root and ends in one of the leaves). Each node at distance t from the root represents a state q_t traversed in a particular sequence at step t. The tree structure is built iteratively by the List Viterbi algorithm. In the first step the tree height is 1 and the leaves represent all possible initial states given the observation sequence $O = (o_1, o_2, \dots, o_T)$, i.e. there are at maximum N leaves at this step. In all other steps t, where $2 \leq t \leq T$, a maximum of K leaves representing a state q_t are added to the tree, i.e. the tree memorizes up to K best paths that end in state q_t. A maximum of $K \cdot N$ leaves are added to the tree at each step. After the tree is built, the $K \cdot N$ leaves are ordered based on their probability and the first K are the top-K solutions we where looking for. The complete sequences are then extracted from the tree by backtracking the paths from the leaves to the root.

Example 6.5. The top-3 results of the keyword query "`people restaurant Naples`" are the ones mapping the keyword `people` into the table *People*, `restaurant` into the table *Restaurant*, and `Naples` into the domains of the attributes *Name Specialty*and *City*, respectively. Both the solutions have the same transition probabilities but different emission probabilities since the likelihood of the mapping of `Naples` into the attribute *City* is higher than the one into *Restaurant* and *Specialty*.

6.6 Related Work

The popularity of keyword searching is constantly increasing. For more than a decade, it has been the successful model in IR for text databases [40] and the web [12]. It has also been adopted by the data management community in the context of XML [17, 26, 43]. To answer a keyword query over XML data, the appearances of the query keywords are first identified within the documents (possibly through some free-text search) and then combined together into Meaningful Lowest Common Ancestor Structures (MLCAS). A score is computed based on this structure and according to this score the respective XML documents containing the MLCAS are ranked and returned to the user. XML benefits from the fact that its basic information model is the "document", which is the same as in IR, thus, many IR techniques can be easily employed in the XML context. On the other hand, keyword search in relational databases is particularly challenging [42], first because the database instances are way larger, and second because the basic model is fundamentally different making hard the identification of the information units that are to be returned. Nevertheless, keyword search over relational data is particularly appealing, and there are already many interesting proposals in the scientific literature [49, 14]. DISCOVER [19] and DBXplorer [2] have been among the first such systems. The typical approach is to build a special index on the contents of the

Algorithm 14: List Viterbi

```
Data: $\lambda = (A, B, \Pi)$ : HMM, $KQ = (k_1, k_2, \dots, k_l)$ : a keyword query, $K$ : top-K
Result: $Q[K][l]$ : top-K state sequences, $P[K][l]$ : top-K state sequences probabilites

Compute_Tree($A$,$B$,$\Pi$,$KQ$,$k$);
begin
    $Tree \leftarrow newnode(root)$ ;
    for state $S_i$ do
        $\delta_0 \leftarrow b_i \cdot \pi_i$ ;
        $appendNode((S_i, \delta_0), Tree)$ ;
    $L[\,] \leftarrow leaves(Tree)$ ;
    for $k_i \in KQ$ do
        array $Tmp[N][K]$;
        for state $S_i$ do
            for $j \in L$ do
                $\delta_k \leftarrow a_{ij} \cdot b_i \cdot \delta_{k-1}$ ;
                $appendNode((S_i, \delta_k), Tmp[S_i][\,])$ ;
            $insert(Tmp[S_i][], Tree)$ ;
            $L[\,] \leftarrow L[\,] + Tmp[S_i][\,]$ ;
            $orderLeafs(L[\,])$;

Backtracking(Tree);
begin
    $t \leftarrow l$;
    for $j \in L$ do
        $Q[j][t] \leftarrow state(L[j])$;
        $P[j][t] \leftarrow probability(L[j])$;
        for $t = l-1, l-2, 1$ do
            $Q[j][t] \leftarrow state(parent(Q[j][t+1], Tree))$;
            $P[j][t] \leftarrow probability(parent(Q[j][t+1], Tree))$;
```

database and then to use that index to identify the appearances of the query keywords in the attribute values. Many approaches use inverted indexes [1, 19, 39] for that purpose, while others, like DBXplore, use symbol tables. A symbol table is an index consisting of a set of triples ⟨*value, attribute, relation*⟩, used to map every value to its schema information. Once the keywords are located, the different ways by which their respective tuples are connected are discovered, forming the so-called "joining network" of tuples or tuple trees, that often become the information unit returned to the user. The joining networks are constructed either directly from the instances, or by building query expressions and evaluating them [19]. DISCOVER is interested in finding total and minimal joining networks. A joining network is total if each keyword query is contained in at least one tuple of the network, and minimal if the removal of any tuple makes the network no longer total. More recent approaches [36], are oriented towards reducing the number of tuples that need to be considered in order to improve previous techniques. BANKS [1] follows a similar approach but employs the Steiner tree algorithm to discover how the tuples are associated. SQAK [42] is another system that is based on the same generic principles but focuses on the discovery of aggregate SQL expressions that describe the intended keyword query semantics. Since the set of possible answers may be large, and since the results are already ranked, the above systems typically return the top-*k* results.

As happened in all the above approaches, the two methods that we presented in this chapter are also trying to identify the possible semantics (i.e., expected answers) to the ambiguous keywords queries, and rank the results. The fundamental difference is that they do not assume any a-priori access to the database instance. Unavoidably, the approaches are based on schema and meta-data, i.e., the intensional information, which makes them applicable to scenarios where the other techniques cannot work. Nevertheless, presented approaches should not be seen as an alternative to the above methods. Since they operate on different information, i.e., the meta-data, they can be used to enhance these techniques by providing a better exploitation of the meta-information and the relationships among the keywords. The idea of using schema information and keyword semantics has been considered in one of the approaches [27], but this is only limited to word semantics based on WordNet. The presented two approaches go further by combining not only additional semantic similarity techniques, similar to those used in schema matching [37], but also on syntactic and structural information. Data behavior has also been considered [42]. All these works are complementary.

Another distinction is on the relationship among the keywords. Existing approaches compute the relationship among the keywords by considering the relationships of the data values in the database instance. It is believed by many that the keyword query should be the one driving the translation and not the data. The presented aproaches take into consideration in a systematic way the position of the keywords in the query itself and the inter-dependencies between the matchings of the keywords to the database structure. Furthermore, they map the keywords to schema elements that will form the SQL query, instead of mapping them to the tuples themselves [42]. This allows them, without performing any query evaluation, to present these queries to the user and communicate the considered semantics. Of course, this is an optional step, but in highly heterogeneous environments, this is the norm for matching and mapping techniques [33].

A related field is keyword disambiguation where several approaches have been developed. For example, in [4] an incremental technique based on WordNet and Description Logics is proposed, in [46] context and ontologies are exploited for removing ambiguity, and in [34] attribute disambiguation enables faceted searches.

With specific reference to the KEYRY approach, a large number of techniques based on HMM that have been proposed typically addresses the following three basic problems [3]: 1) *finding the MLSS,* i.e. given a HMM model λ and a sequence of observations O_l, find the correspondent state sequence Q_l that has the highest probability of generating O_l; 2) *evaluation problem,* i.e. given a model λ and a sequence of observations O_l, evaluate the probability that the model λ has generated the observations O_l and 3) *learning the parameters,* i.e. given a training dataset of observation sequences O, learn the model λ that maximizes the probability of generating O. This chapter described a way to apply the List Viterbi algorithm for addressing the first problem, i.e. decoding the HMM by computing the top-K answers to a keyword query. Other interesting applications of List Viterbi have been proposed especially in the data transmission field, where it has been exploited for detecting the correct message in cases of transmission errors [38], and for source-channel coding of images [20].

In [23], the Baum-Welch, the Viterbi training, and the Monte Carlo EM training, are described and an extension of the latter two approaches is proposed in order to make them more efficient. Since these algorithms can in practice be slow and computationally expensive, several techniques have been proposed for improving the learning under particular conditions: [24] proposes an "adjusted Viterbi training" with the same complexity level as the Viterbi training algorithm, but computing more accurate results. Differently from these proposals, the presented scenario requires an online training approach in order to take into account the user's feedback as soon as it is received.

Finally, the presented methods can also find applications in the field of graphical tools that assist the user in formulating queries [32]. By finding the different interpretations of a keyword query, one could detect related schema structures, make suggestions and guide the user in the query formulation. Furthermore, in cases of exploratory searches, the user can use the generated interpretations as a way to explore an (unknown) data source and understand better its semantics.

6.7 Conclusion

The chapter presented an overview of two methods for translating keyword queries over a relational database into SQL so that they can be executed. The main focus was in a situation in which the contents of the database is unknown and the only information available is the meta-information, i.e., schema and constraints. In both cases the idea was to understand the semantics of the keywords in order to correctly map them to database structures and then use this mapping to guide the query generation process. The first method was an extension of the Hungarian algorithm and the second of the Viterbi algorithms. Preliminary experiments have shown that both approaches work equally well, however, a clear understanding on when each one performs better is still not clear.

References

[1] Aditya, B., Bhalotia, G., Chakrabarti, S., Hulgeri, A., Nakhe, C., Parag, Sudarshan, S.: Banks: Browsing and keyword searching in relational databases. In: VLDB, pp. 1083–1086. Morgan Kaufmann (2002)

[2] Agrawal, S., Chaudhuri, S., Das, G.: Dbxplorer: A system for keyword-based search over relational databases. In: ICDE, pp. 5–16. IEEE Computer Society

(2002)
[3] Alpaydin, E.: Introduction to Machine Learning, second edition. The MIT Press (2010)
[4] Bergamaschi, S., Bouquet, P., Giacomuzzi, D., Guerra, F., Po, L., Vincini, M.: An incremental method for the lexical annotation of domain ontologies. Int. J. Semantic Web Inf. Syst. **3**(3), 57–80 (2007)
[5] Bergamaschi, S., Domnori, E., Guerra, F., Lado, R.T., Velegrakis, Y.: Keyword search over relational databases: a metadata approach. In: T.K. Sellis, R.J. Miller, A. Kementsietsidis, Y. Velegrakis (eds.) SIGMOD Conference, pp. 565–576. ACM (2011)
[6] Bergamaschi, S., Domnori, E., Guerra, F., Orsini, M., Lado, R.T., Velegrakis, Y.: Keymantic: Semantic keyword-based searching in data integration systems. PVLDB **3**(2), 1637–1640 (2010)
[7] Bergamaschi, S., Guerra, F., Rota, S., Velegrakis, Y.: A hidden markov model approach to keyword-based search over relational databases. In: to appear in ER. Springer (LNCS) (2011)
[8] Bergamaschi, S., Sartori, C., Guerra, F., Orsini, M.: Extracting relevant attribute values for improved search. IEEE Internet Computing **11**(5), 26–35 (2007)
[9] Bergman, M.K.: The deep web: Surfacing hidden value. Journal of Electronic Publishing **7**(1) (2001). URL `http://dx.doi.org/10.3998/3336451.0007.104`
[10] Bleiholder, J., Naumann, F.: Data fusion. ACM Comput. Surv. **41**(1) (2008)
[11] Bourgeois, F., Lassalle, J.C.: An extension of the Munkres algorithm for the assignment problem to rectangular matrices. Communications of ACM **14**(12), 802–804 (1971)
[12] Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Computer Networks **30**(1-7), 107–117 (1998)
[13] Burkard, R., Dell'Amico, M., Martello, S.: Assignment Problems. SIAM Society for Industrial and Applied Mathematics, Philadelphia (2009)
[14] Chakrabarti, S., Sarawagi, S., Sudarshan, S.: Enhancing search with structure. IEEE Data Eng. Bull. **33**(1), 3–24 (2010)
[15] Cilibrasi, R., Vitányi, P.M.B.: The google similarity distance. IEEE Transactions on Knowledge & Data Engineering **19**(3), 370–383 (2007)
[16] Cohen, W.W., Ravikumar, P.D., Fienberg, S.E.: A comparison of string distance metrics for name-matching tasks. In: IIWeb, pp. 73–78 (2003)
[17] Florescu, D., Kossmann, D., Manolescu, I.: Integrating keyword search into xml query processing. In: BDA (2000)
[18] Haofen Wang Kang Zhang, Q.L., Tran, D.T., Yu, Y.: Q2semantic: A lightweight keyword interface to semantic search. In: Proceedings of the 5th European Semantic Web Conference, Tenerife, Spain, pp. 584–598. LNCS (2008)
[19] Hristidis, V., Papakonstantinou, Y.: Discover: Keyword search in relational databases. In: VLDB, pp. 670–681 (2002)

[20] Konstanz, U., Rder, M., Rder, M., Hamzaoui, R., Hamzaoui, R.: Fast list viterbi decoding and application for source-channel coding of images. In: Konstanzer Schriften in Mathematik und Informatik, http://www.inf.uni-konstanz.de/ Preprints/preprints-all.html, pp. 801–804 (2002)

[21] Kotidis, Y., Marian, A., Srivastava, D.: Circumventing Data Quality Problems Using Multiple Join Paths. In: CleanDB (2006)

[22] Kumar, R., Tomkins, A.: A Characterization of Online Search Behavior. IEEE Data Engineering Bulletin **32**(2), 3–11 (2009)

[23] Lam, T.Y., Meyer, I.M.: Efficient algorithms for training the parameters of hidden markov models using stochastic expectation maximization (em) training and viterbi training. Algorithms for Molecular Biology **5**(38) (2010). DOI 10.1186/1748-7188-5-38

[24] Lember, J., Koloydenko, A.: Adjusted viterbi training. Probab. Eng. Inf. Sci. **21**, 451–475 (2007). DOI 10.1017/S0269964807000083. URL `http://portal.acm.org/citation.cfm?id=1291117.1291125`

[25] Li, L., Shang, Y., Shi, H., Zhang, W.: Performance evaluation of hits-based algorithms. In: Communications, Internet, and Information Technology, pp. 171–176 (2002)

[26] Li, Y., Yu, C., Jagadish, H.V.: Schema-free XQuery. In: VLDB, pp. 72–83 (2004)

[27] Liu, F., Yu, C.T., Meng, W., Chowdhury, A.: Effective keyword search in relational databases. In: SIGMOD, pp. 563–574. ACM (2006)

[28] Madhavan, J., Ko, D., Kot, L., Ganapathy, V., Rasmussen, A., Halevy, A.: Google's deep web crawl. Proceeding of the Very Large DataBases (VLDB) Endow. **1**(2), 1241–1252 (2008). DOI http://doi.acm.org/10.1145/1454159.1454163. URL `http://portal.acm.org/citation.cfm?id=1454163`

[29] Maier, D., Ullman, J.D., Vardi, M.Y.: On the Foundations of the Universal Relation Model. ACM Trans. Database Syst. **9**(2), 283–308 (1984)

[30] Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: ICDE, pp. 117–128. IEEE Computer Society (2002)

[31] Mena, E.: OBSERVER: An Approach for Query processing in Global Information Systems based on Interoperation across Pre-exisiting Ontologies. University of Zaragoza (1998)

[32] Nandi, A., Jagadish, H.V.: Assisted querying using instant-response interfaces. In: SIGMOD, pp. 1156–1158. ACM (2007)

[33] Popa, L., Velegrakis, Y., Miller, R.J., Hernandez, M.A., Fagin, R.: Translating web data. In: VLDB, pp. 598–609 (2002)

[34] Pound, J., Paparizos, S., Tsaparas, P.: Facet discovery for structured web search: a query-log mining approach. In: SIGMOD Conference, pp. 169–180. ACM (2011)

[35] Pu, K.Q.: Keyword query cleaning using hidden markov models. In: M.T. Özsu, Y. Chen, L.C. 0002 (eds.) KEYS, pp. 27–32. ACM (2009)

[36] Qin, L., Yu, J.X., Chang, L.: Keyword search in databases: the power of rdbms. In: SIGMOD, pp. 681–694. ACM (2009)

[37] Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB Journal **10**(4), 334–350 (2001)

[38] Seshadri, N., Sundberg, C.E.: List Viterbi decoding algorithms with applications. Communications, IEEE Transactions on **42**(234), 313 –323 (1994). DOI 10.1109/TCOMM.1994.577040

[39] Simitsis, A., Koutrika, G., Ioannidis, Y.E.: Précis: from unstructured keywords as queries to structured databases as answers. VLDB Journal **17**(1), 117–149 (2008)

[40] Singhal, A., Buckley, C., Mitra, M.: Pivoted document length normalization. In: SIGIR, pp. 21–29 (1996)

[41] Tata, S., Lohman, G.M.: Sqak: doing more with keywords. In: J.T.L. Wang (ed.) Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, pp. 889–902. ACM (2008)

[42] Tata, S., Lohman, G.M.: SQAK: doing more with keywords. In: SIGMOD, pp. 889–902. ACM (2008)

[43] Theobald, M., Bast, H., Majumdar, D., Schenkel, R., Weikum, G.: TopX: efficient and versatile top-k query processing for semistructured data. VLDB Journal **17**(1), 81–115 (2008)

[44] Tran, T., Mathäß, T., Haase, P.: Usability of keyword-driven schema-agnostic search. In: 7th Extended Semantic Web Conference (ESWC'10), Greece. Springer (2010)

[45] Tran, T., Wang, H., Rudolph, S., Cimiano, P.: Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In: ICDE, pp. 405–416. IEEE Computer Society (2009). DOI http://dx.doi.org/10.1109/ICDE.2009.119

[46] Trillo, R., Gracia, J., Espinoza, M., Mena, E.: Discovering the semantics of user keywords. J. UCS **13**(12) (2007)

[47] Wright, A.: Searching the deep web. Communications of The ACM **51**, 14–15 (2008). DOI 10.1145/1400181.1400187

[48] Yu, J.X., Qin, L., Chang, L.: Keyword Search in Databases. Morgan & Claypool (2010)

[49] Yu, J.X., Qin, L., Chang, L.: Keyword Search in Databases. Synthesis Lectures on Data Management. Morgan & Claypool Publishers (2010)

[50] Zenz, G., Zhou, X., Minack, E., Siberski, W., Nejdl, W.: From keywords to semantic queries-incremental query construction on the semantic web. Journal of Web Semantics **7**(3), 166–176 (2009). DOI http://dx.doi.org/10.1016/j.websem.2009.07.005