SoftFacts: A Top-k Retrieval Engine for Ontology Mediated Access to Relational Databases

Umberto Straccia
ISTI - CNR
Pisa, ITALY
straccia@isti.cnr.it

***Abstract*—We outline SoftFacts, an ontology mediated top-k information retrieval system over relational databases. An ontology layer is used to define (in terms of a OWL-QL like Semantic Web language) the relevant abstract concepts and relations of the application domain, while graded facts are stored into a relational database. Queries are conjunctive queries with ranking aggregates and scoring functions and the results of a query may be ranked according to user defined scoring functions. We will illustrate SoftFacts' architecture, the representation and the query language, sketch the reasoning algorithms of the SoftFacts system and its Protégé plug-in.**

***Index Terms*—Knowledge-based Systems, Ontologies, Fuzzy Description Logics, Top-k retrieval.**

I. Introduction

Description Logics (DLs) [2] have gained popularity due to their application in the context of the *Semantic Web*. DLs play a particular role as they are essentially the theoretical counterpart of state of the art Semantic Web ontology languages of the *OWL* family [1].

Clearly, in the Semantic Web context, data is typically very large and dominate the intentional level of the ontologies. Hence, while in the above mentioned contexts one could still accept reasoning that is exponential on the intentional/ontological part, it is mandatory that reasoning is polynomial in the data size, *i.e.* in *data complexity* [24]. Recently efficient management of large amounts of data and its computational complexity analysis has become a primary concern of research in DLs and in ontology reasoning systems [1], [4], [8], [14], [15].

In this paper, we describe the salient features of the SoftFacts system [2], whose aim is to allow an ontology mediated access to relational databases for data intensive applications. Informally, data is stored into a database and a DL is used to define the relevant abstract concepts and relations of the application domain (see Fig. 1).

Main features of the SoftFacts system are: (i) The SoftFacts ontology language belongs to the family of DLR-Lite [4]. DLR-Lite is different from usual DLs as it supports n-ary relations ($n \geqslant 1$), whereas DLs support usual unary relations (called *concepts*) and binary relations (called *roles*) only. The language is closely related to *OWL QL* (a so-called profile of the web ontology language *OWL 2*) [3]. In fact, *OWL QL* can be managed by SoftFacts; (ii) At the query level, we consider Datalog like conjunctive queries with ranking aggregates and scoring functions [10], [11]. The results of a query may be ranked according to some scoring function. In fact, SoftFacts supports *Top-k Query Answering* [12], [18], [19], [20], [21], (find top-k scored tuples satisfying query), *e.g.* "find cheap hotels close to the train station", where cheap and price are an user defined function of the distance and price, respectively. (iii) SoftFacts provides also an integrated graphical interface within the well-known Semantic Web ontology editor Protégé [4] (see Fig. 2).

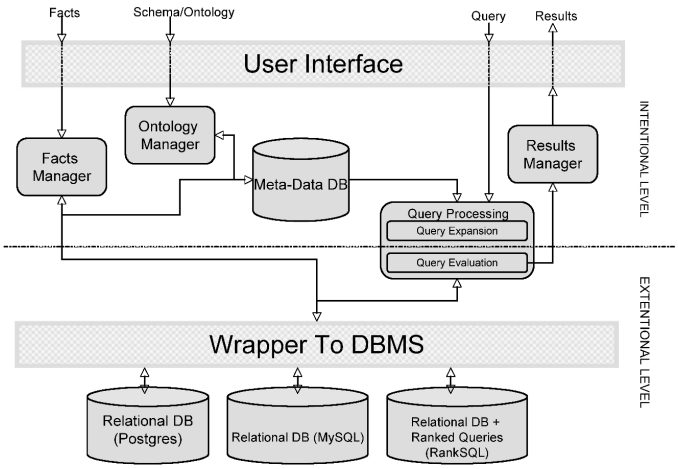


Fig. 1. Architecture.

Related work: While there are many works addressing the top-k problem for vague queries over databases (cf. [6], [7], [9], [10], [11]), little is known for the corresponding problem in knowledge representation and reasoning (see [22] for a survey), as *e.g.* [12], [16], [17], [18], [19], [20], [21], [25]. We also refer the reader to [5] for a crisp, main memory, DL-Lite implementation and application, and to [13] for a fuzzy DL-Lite language in the spirit of [20] (and less expressive language than ours) but not addressing specifically the top-k retrieval problem and ranking aggregates. However, to the best of our knowledge, SoftFacts is the unique system based on a OWL-like Semantic Web language, fully supporting top-k retrieval, and interfacing with several databases.

In the following, we will illustrate SoftFacts' architecture,

[1] http://www.w3.org/2004/OWL/.
[2] See, http://www.straccia.info/software/SoftFacts/SoftFacts.html
[3] http://www.w3.org/TR/owl2-profiles/.
[4] http://protege.stanford.edu/

978-1-4244-6587-3/10/$25.00 É2010 IEEE

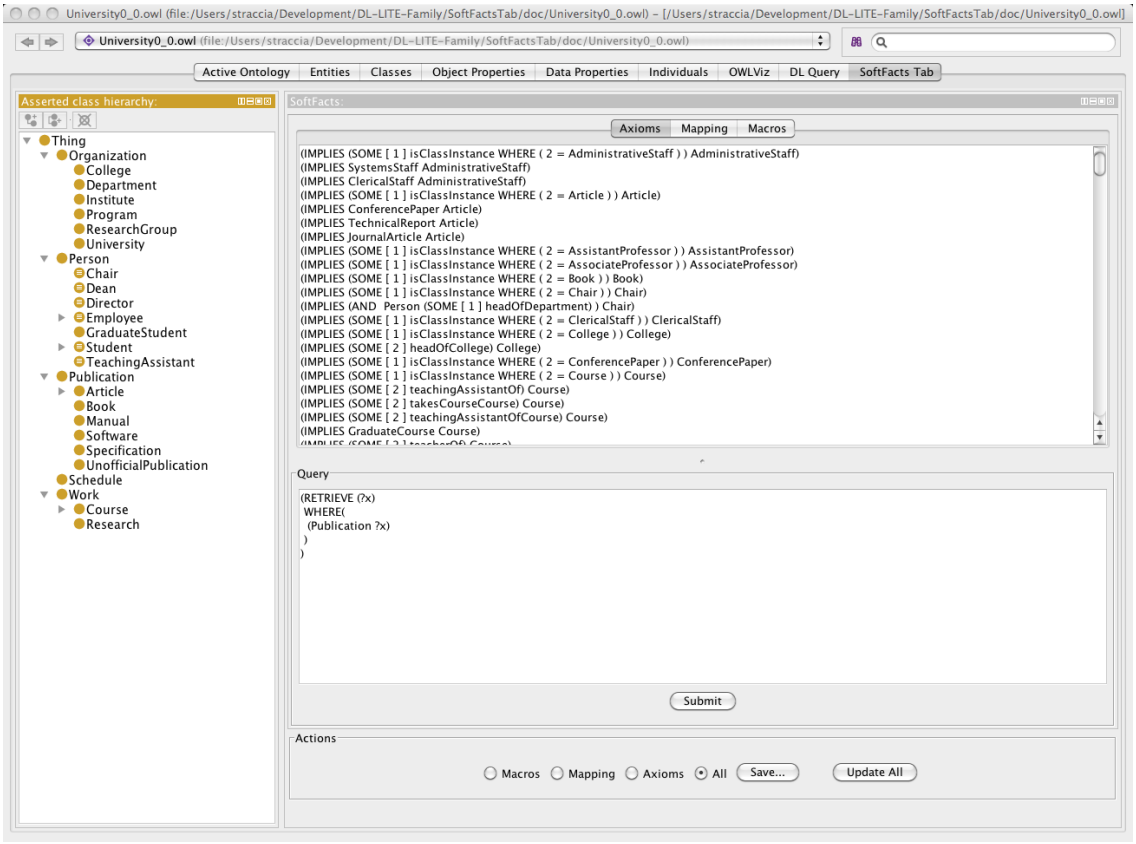


Fig. 2. SoftFacts Protégé plug-in.

the representation and the query language, sketch the reasoning algorithms of the SoftFacts system and its Protégé plug-in.

II. The architecture

The SoftFacts architecture has two basic components: the DL-based ontology component (*the intentional level*) and the database component (*the extensional level*) as shown in Fig. 1. The *DL-component* supports both the definition of the ontology and query answering. In particular, it provides a logical query and representation language, which is an extension of the DL language DLR-Lite [4], [18], [20] and supports ranking queries. Concerning the *database component*, SoftFacts supports access to various different database systems. The access to these systems is transparent as managed by an appropriate wrapper.

Operationally, a user submits a conceptual query (a Datalog like logic-based conjunctive query with a scoring atom) by means of the DL-component. The DL-component will then use the ontology to *reformulate* the initially query into one or several queries (query expansion, for details see [23]). These queries are then translated and submitted to the underlying database system (using the wrapper). The database system then provides back the top-k answers for each of the issued queries. The ranked lists will then be merged into one final result list and displayed to the user (using the the query evaluation module).

III. The representation and query language

The logic SoftFacts adopts is based on a fuzzy extension of the DLR-Lite [4] DL without negation. DLR-Lite is an extension of DL-Lite (which is the logic behind *OWL QL* (a profile of the web ontology language *OWL 2* [5]) as it supports n-ary relations whereas DLs support usual unary relations (called *concepts*) and binary relations (called *roles*) only. Please note that SoftFacts does not consider negation of atoms, as supported in DL-Lite, as these do not play any role at query answering time, the main task supported by SoftFacts. Negated atoms play a role only at knowledge base consistency checking time. In SoftFacts, any knowledge base is consistent. Therefore, to what concerns query answering, the drop of negation is harmless.

A. The representation language

A *knowledge base* $\mathcal{K} = \langle \mathcal{F}, \mathcal{O}, \mathcal{A} \rangle$ consists of a *facts component* $\mathcal{F}$, an *Ontology component* $\mathcal{O}$ and an *abstraction component* $\mathcal{A}$, which are defined as follows (for a detailed account of the semantics, see [23]).

Facts Component. $\mathcal{F}$ is a finite set of expressions of the form

$$R(c_1, \ldots, c_n)[s] \ ,$$

where R is an n-ary relation, every c_i is a constant, and s is a degree of truth (or *score*) in $[0, 1]$ indicating to which extent the

[5] http://www.w3.org/TR/2009/REC-owl2-overview-20091027/.

Profile							
profID	FirstName	LastName	BirthDate	CityOfBirth	Address	City	...
2	Wayne	Hernandez	1979-10-04	Berlin	Via Volta	Terni	...
34	Hillary 156	Gadducci	1978-01-27	Bangalore	Church ST	New York	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Fig. 3. The profile table.

tuple $\langle c_1, \ldots, c_n \rangle$ is an instance of relation R. Facts are stored in a relational database. We may omit the score component and in such case the value 1 is assumed.

Example 1 ([17]): Suppose we have Curricula Vitæ. Some basic information is stored into the Profile relation. An excerpt is shown in Fig. 3. □

Note the difference to so-called fuzzy relational databases (see, *e.g.* [3], [26]), in which values of a column may be fuzzy (*e.g.*, the person's age is "young"), whereas this is not the case here.

Ontology Component. The ontology component is used to define the relevant abstract concepts and relations of the application domain by means of axioms. Specifically, $\mathcal{O}$ is a finite set of *axioms* having the form

$$Rl_1 \sqcap \ldots \sqcap Rl_m \sqsubseteq Rr \ ,$$

where $m \geqslant 1$, all Rl_i and Rr have the same arity and each Rl_i is a so-called *left hand relation* and Rr is a *right hand relation* (note that recursive axioms are allowed). We assume that relations occurring in $\mathcal{F}$ do not occur in axioms (so, we do not allow that database relation names occur in $\mathcal{O}$). The intuitive semantics is that if a tuple $\mathbf{c}$ is instance of each relation Rl_i to degree s_i then $\mathbf{c}$ is instance of Rr to degree $\min(s_1, \ldots, s_m)$.

As illustrative purpose, a simple ontology axiom may be of the form

$$\mathsf{ItalianCity} \sqsubseteq \mathsf{EuropeanCity}$$

with informal reading "any Italian city is an European city". Here ItalianCity and EuropeanCity are unary relations (*i.e.*, concepts) with signature ItalianCity(id) and EuropeanCity(id), respectively. Similarly, the ontology axiom

$$\mathsf{ItalianCity} \sqcap \mathsf{BigCity} \sqsubseteq \mathsf{BigEuropeanCity}$$

has informal reading "any Italian city, which is also big, is a big European city" (here BigCity and BigEuropeanCity are again unary relations (concepts) with signature BigCity(id) and BigEuropeanCity(id), respectively.

The exact syntax of the relations appearing on the left-hand and right hand side of ontology axioms is specified below (where $h \geqslant 1$):

$$\begin{array}{lcl} Rr & \longrightarrow & A \mid R[i_1, \ldots, i_k] \\ Rl & \longrightarrow & A \mid R[i_1, \ldots, i_k] \mid \\ & & R[i_1, \ldots, i_k].(Cond_1, \ldots, Cond_h) \\ Cond & \longrightarrow & ([i] \leqslant v) \mid ([i] < v) \mid ([i] \geqslant v) \mid ([i] > v) \mid \\ & & ([i] = v) \mid ([i] \neq v) \end{array}$$

where A is an *atomic concept* (an unary predicate), R is an n-ary relation with $n \geqslant 2$, $1 \leqslant i_1, i_2, \ldots, i_k \leqslant n$, $1 \leqslant i \leqslant n$ and v is a value of the appropriate type. Here $R[i_1, \ldots, i_k]$ is the projection of the relation R on the columns $i_1, \ldots, i_k$ (the order of the indexes matters). Hence, $R[i_1, \ldots, i_k]$ has arity k. For instance, Profile$[1, 3]$ is the binary relation that is the projection on the first and third column of the Profile relation. Note that Profile$[1, 3]$ is different from Profile$[3, 1]$ (columns are inverted).

$$\begin{array}{rcll} \mathsf{Legislators} & \sqsubseteq & \mathsf{Legal\_Support\_Workers} & (1) \\ \mathsf{Auditors} & \sqsubseteq & \mathsf{Accountants\_and\_Auditors} & (2) \\ \mathsf{Bakers} & \sqsubseteq & \mathsf{Food\_Processing\_Workers} & (3) \\ \mathsf{knowsLanguage}[2] & \sqsubseteq & \mathsf{Language} & (4) \\ \mathsf{hasDegree}[2] & \sqsubseteq & \mathsf{Degree} & (5) \end{array}$$

Fig. 4. Excerpt of a CV ontology.

On the other hand, $R[i_1, \ldots, i_k].(Cond_1, \ldots, Cond_l)$ further restricts the projection $R[i_1, \ldots, i_k]$ according to the conditions specified in $Cond_i$. For instance, $([i] \leqslant v)$ specifies that the values of the i-th column have to be less or equal than the value v and similarly, for the other conditions. We assume that the comparison occurs among values with a comparable type. For instance,

$$\mathsf{Profile}[1, 5].(([5] \geqslant 1979))$$

corresponds to the set of tuples $\langle \mathsf{profID}, \mathsf{BirthDate} \rangle$ such that the fifth column of the relation Profile, *i.e.* the person's birth date, is equal or greater than 1979.

Example 2 (Example 1 cont.): An excerpt of the domain ontology is described in Fig. 4 and partially encodes an ontology used to describe Curricula Vitæ. We assume that we have relations hasDegree(profID, degID, marks), knowsLanguage(profID, lanID) and an atomic concept Degree(degID). For instance, axiom (4) states that the languages known by profile profID should be languages. □

Abstraction Component. The abstraction component (similarly to [5], [15]) is a set of "abstraction statements" that allow to connect atomic concepts and relations to physical relational tables. Essentially, this component is used as a wrapper to the underlying database and, thus, prevents that relational table names occur in the ontology. As illustrative purpose, assume that we have a relation Jobs in a database with signature Jobs(jobID, name), where the first column is of type int, while the second is of type string. Then, an example of abstraction statement is

$$\mathsf{jobName} \mapsto \mathsf{Jobs}(\mathsf{jobID}[\mathsf{int}], \mathsf{name}[\mathsf{string}]) \ ,$$

by means of which we state that the relation jobName occurring in the ontology component, has arity two and has to be mapped into the relation Jobs occurring in the database.

Formally, let R be a relation symbol and let T be an m-ary table in the database. Let $c_1, \ldots, c_n$ be column names of table T each of which of type t_i ($n \leqslant m$). We assume that R occurs in $\mathcal{O}$, while T occurs in $\mathcal{F}$. We also assume that the score of an instance of T is stored in the column c_{score}. Then

an *abstraction statement* is of the form

$$R \mapsto T(c_1[t_1], \ldots, c_n[t_n])[c_{score}] \ ,$$

stating that R is an n-ary relation of the ontology component, that is mapped into the projection on columns $c_1 \ldots, c_n$ of table T. The score of the tuples of R is provided by column c_{score} of table T. The score column c_{score} may be omitted and in that case the score 1 is assumed for the tuples. Finally, we assume that there is at most one abstraction statement for each abstract relational symbol R.

B. The query language

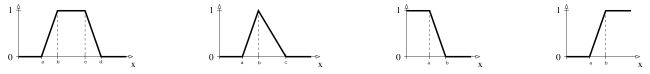
Concerning queries, a *query* consists of a "conjunctive query", with a scoring function to rank the answers. We first present *ranking queries* such as presented in [12] and then add to them ranking aggregates [10].

A *ranking query* [12] is of the form

$$\begin{aligned} q(\mathbf{x})[s] \quad \leftarrow \quad & \exists \mathbf{y} \ R_1(\mathbf{z}_1)[s_1], \ldots, R_l(\mathbf{z}_l)[s_l], \\ & \mathsf{OrderBy}(s = f(s_1, \ldots, s_l, p_1(\mathbf{z}'_1), \ldots, p_h(\mathbf{z}'_h))), \\ & \mathsf{Limit}(k) \end{aligned}$$

where

1) q is an n-ary relation, every R_i is an n_i-ary relation. $R_i(\mathbf{z_i})$ may also be of the form $(z \leqslant v), (z < v), (z \geqslant v), (z > v), (z = v), (z \neq v)$, where z is a variable, v is a value of the appropriate concrete domain;
2) $\mathbf{x}$ are the *distinguished variables*.
3) $\mathbf{y}$ are existentially quantified variables called the *non-distinguished variables*. We omit to write $\exists \mathbf{y}$ when $\mathbf{y}$ is clear from the context;
4) $\mathbf{z_i}, \mathbf{z'_j}$ are tuples of constants or variables in $\mathbf{x}$ or $\mathbf{y}$;
5) $s, s_1, \ldots, s_l$ are distinct variables and different from those in $\mathbf{x}$ and $\mathbf{y}$;
6) p_j is an n_j-ary *fuzzy predicate* assigning to each n_j-ary tuple $\mathbf{c}_j$ a *score* $p_j(\mathbf{c}_j) \in [0, 1]$. Such predicates are called *expensive predicates* in [6] as the score is not pre-computed off-line, but is computed on query execution. We require that an n-ary fuzzy predicate p is *safe*, that is, there is not an m-ary fuzzy predicate p' such that $m < n$ and $p = p'$. Informally, all parameters are needed in the definition of p. Note that concerning fuzzy predicates, we may use the so-called left shoulder, right shoulder, triangular and trapezoidal functions (see Fig. 5), which are well known fuzzy membership functions in fuzzy set theory;
7) f is a *scoring* function $f\colon ([0, 1])^{l+h} \rightarrow [0, 1]$, which combines the scores of the l relations $R_i(\mathbf{c}'_i)$ and the n fuzzy predicates $p_j(\mathbf{c}''_j)$ into an overall *score* to be assigned to the rule head $R(\mathbf{c})$. We assume that f is *monotone*, that is, for each $\mathbf{v}, \mathbf{v}' \in ([0, 1])^{l+h}$ such that $\mathbf{v} \leqslant \mathbf{v}'$, it holds $f(\mathbf{v}) \leqslant f(\mathbf{v}')$, where $(v_1, \ldots, v_{l+h}) \leqslant (v'_1, \ldots, v'_{l+h})$ iff $v_i \leqslant v'_i$ for all i. We also assume that the computational cost of f and all fuzzy predicates p_i is bounded by a constant;
8) $\mathsf{Limit}(k)$ indicates the number of answers to retrieve and is optional. If omitted, all answers are retrieved.



(a) (b) (c) (d)

Fig. 5. (a) Trapezoidal function $trz(x; a, b, c, d)$, (b) triangular function $tri(x; a, b, c)$, (c) left shoulder function $ls(x; a, b)$, and (d) right shoulder function $rs(x; a, b)$.

HasDegree		
profID	degID	Mark
2	29	107
34	25	104
⋮	⋮	⋮

Degree	
degID	Name
29	Civil_Structural_Engineering
25	Chemical_Engineering
⋮	⋮

Fig. 6. The HasDegree and Degree tables.

We call $q(\mathbf{x})[s]$ its *head*, $\exists \mathbf{y}.R_1(\mathbf{z}_1)[s_1], \ldots, R_l(\mathbf{z}_l)[s_l]$ its *body* and $\mathsf{OrderBy}(s = f(s_1, \ldots, s_l, p_1(\mathbf{z}'_1), \ldots, p_h(\mathbf{z}'_h))$ the *scoring atom*. We also allow the scores $[s], [s_1], \ldots, [s_l]$ and the scoring atom to be omitted. In this case we assume the value 1 for s_i and s instead.

The informal meaning of such a query is: if $\mathbf{z}_i$ is an instance of R_i to degree at least or equal to s_i, then $\mathbf{x}$ is an instance of q to degree at least or equal to s, where s has been determined by the scoring atom. Example queries are:

$\mathsf{q(x)} \leftarrow \mathsf{SportsCar(x)}$
// find sports cars

$\mathsf{q(x)} \leftarrow \mathsf{SportsCar(x)}, \mathsf{hasSpeed(x, y)}, (\mathsf{y} \geqslant 240)$
// find sports cars whose speed exceed 240

$\mathsf{q(x)[s]} \leftarrow \mathsf{SportsCar(x)[s_1]}, \mathsf{hasPrice(x, p)}, \mathsf{OrderBy(s} = 0.7 \cdot \mathsf{s_1} + 0.3 \cdot ls(\mathsf{p}; 10000, 14000))$
// find cheap sports cars

Note how in the last query we combine the degree of being sports car with the degree of being the car "cheap", where this latter is defined by means of the left-shoulder function $ls(p; 10000, 14000)$.

Example 3 (Example 2 cont.): Assume that we have the two relational tables as in Fig. 6. Assume that we have also the abstract mappings

$$\begin{aligned} \mathsf{hasName} &\mapsto \mathsf{Profile(lastName[string])} \\ \mathsf{hasDegree} &\mapsto \mathsf{HasDegree(profID[int], degID[int])} \\ \mathsf{hasMark} &\mapsto \mathsf{HasDegree(profID[int], mark[int])} \\ \mathsf{hasDegreeName} &\mapsto \mathsf{Degree(degID[int], name[string])} \ . \end{aligned}$$

A query searching for CV's with a degree with mark between 100 (minimum) and 110 (maximum) can be expressed as

$$\begin{aligned} \mathsf{q(id, name, degree, mark)[s]} \quad \leftarrow \quad & \mathsf{CV(id)}, \mathsf{hasName(id, name)}, \mathsf{hasDegree(id, y)}, \\ & \mathsf{hasDegreeName(y, degree)}, \mathsf{hasMark(id, mark)}, \\ & \mathsf{OrderBy(s} = \mathsf{rs(mark}; 100, 110)) \end{aligned}$$

Then, we may have the results

id	name	degree	mark	score
⋮	⋮	⋮	⋮	⋮
2	Hernandez	Civil_Structural_Engineering	107	0.7
⋮	⋮	⋮	⋮	⋮
34	Gadducci	Chemical_Engineering	104	0.4
⋮	⋮	⋮	⋮	⋮

□

As next, we extend the query language by allowing so-called ranking aggregates to occur in a query [10]. Essentially, ranking aggregates apply usual SQL aggregate functions such as $\mathsf{SUM}, \mathsf{AVG}, \mathsf{MAX}, \mathsf{MIN}$ to the scores of group of tuples, which are answers to a query. For instance, by referring to Example 3, a person (*e.g.*, Gadducci) may held more than one degree and we would like to rank a CV with more degrees better than one with just one (*e.g.* we may would like to *sum-up* the scores of all degrees of Gadducci).

Example 4 (Example 3 cont.): Assume that we additionally would like to sum-up the scores of the degrees of each person and find the top-k ranked tuples. Then, such a query may be expressed as

$$\begin{aligned} \mathsf{q}(\mathsf{id},\mathsf{name})[\mathsf{s}] \quad \leftarrow \quad & \mathsf{CV}(\mathsf{id}), \mathsf{hasName}(\mathsf{id},\mathsf{name}), \mathsf{hasMark}(\mathsf{id},\mathsf{mark}), \\ & \mathsf{GroupedBy}(\mathsf{id},\mathsf{name}), \\ & \mathsf{OrderBy}(\mathsf{s} = \mathsf{SUM}[\mathsf{rs}(\mathsf{mark};100,110)]), \\ & \mathsf{Limit}(k) \end{aligned}$$

Intuitively, for the above query, we ask to group all tuples according to $\langle \mathsf{id}, \mathsf{name} \rangle$ and then for each group to sum-up the scores. That is, if $g = \{t_1, \ldots, t_n\}$ is a group of tuples with same id and name, where each tuple has score s_i computed as $\mathsf{rs}(\mathsf{mark}; 100, 110)$ then the score s_g of the group g is $\sum_{t_i} s_i$. A group g is ranked then according to its score s_g and the top-k ranked groups are returned. □

Formally, let @ be ranking aggregate function with @ $\in \{\mathsf{SUM}, \mathsf{AVG}, \mathsf{MAX}, \mathsf{MIN}\}$ then a query with ranking aggregates is of the form

$$\begin{aligned} q(\mathbf{x})[s] \quad \leftarrow \exists \mathbf{y} \quad & R_1(\mathbf{z}_1)[s_1], \ldots, R_l(\mathbf{z}_l)[s_l], \mathsf{GroupedBy}(\mathbf{w}), \\ & \mathsf{OrderBy}(s = @[f(s_1, \ldots, s_l, p_1(\mathbf{z}'_1), \ldots, p_h(\mathbf{z}'_h)]), \\ & \mathsf{Limit}(k) \end{aligned}$$

where additionally $\mathbf{w}$ is a list of variables according to which we want to group the tuples and @ is the aggregate function according to which to compute the score of the group. $\mathsf{GroupBy}(\mathbf{w})$ is called the *grouping atom* and we assume that $\mathbf{w}$ are variables in $\mathbf{x}$ or $\mathbf{y}$ such that each variable in $\mathbf{x}$ occurs in $\mathbf{w}$.

Finally, given a knowledge base $\mathcal{K}$, and a query q, the *top-k retrieval problem* is the problem to retrieve k tuples $\langle \mathbf{c}, s \rangle$ that instantiate the query relation q with maximal scores (if k such tuples exist), and rank them in decreasing order relative to the score s.

C. Query answering

It is well known in the database community that naive approaches to the top-k retrieval problem (*e.g.* retrieval all records, sort them and cut-out the top-k ones) fail for large size databases. This is true as well for top-k retrieval in logical languages, as shown [23]. So, specific algorithms have to be developed. Specifically, from a query answering point of view, SoftFacts extends the reasoning method [4] to the fuzzy case and is a generalisation of the one described in [4], [18], [20]. Informally, given a query q

1) by considering $\mathcal{O}$, the user query q is *reformulated* into a set of conjunctive queries $r(q, \mathcal{O})$. Roughly, the basic idea is that the reformulation procedure closely resembles a top-down resolution procedure for logic programming, where each axiom is seen as a logic programming rule. For instance, given the query

$$q(x)[s] \leftarrow A(x)[s'], \mathsf{OrderBy}(s = f(s'))$$

and suppose that $\mathcal{O}$ contains the axioms

$$\begin{aligned} B_1 \sqcap B_2 \quad & \sqsubseteq \quad A \\ C_1 \sqcap C_2 \quad & \sqsubseteq \quad A \ , \end{aligned}$$

then we can reformulate the query into two queries

$$\begin{aligned} q(x)[s] \quad & \leftarrow \quad B_1(x)[s_1], B_2(x)[s_2], \mathsf{OrderBy}(s = f(\min(s_1, s_2))) \\ q(x)[s] \quad & \leftarrow \quad C_1(x)[s_1], C_2(x)[s_2], \mathsf{OrderBy}(s = f(\min(s_1, s_2))) \ ; \end{aligned}$$

2) from the set of reformulated queries $r(q, \mathcal{O})$ we remove redundant queries;
3) the reformulated queries $q' \in r(q, \mathcal{O})$ are translated to ranked SQL queries and evaluated. The query evaluation of each ranked SQL query returns the top-k answer set for that query;
4) all the $n = |r(q, \mathcal{O})|$ top-k answer sets are merged into the unique top-k answer set using the *Disjunctive Threshold Algorithm* (DTA, see *e.g.* [20], [23]).

We refer the reader to [23] for details on how the SoftFacts systems addresses the top-k retrieval problem and the experiments we conducted with it. It can be shown that the top-k retrieval problem can be solved in LOGSPACE data complexity.

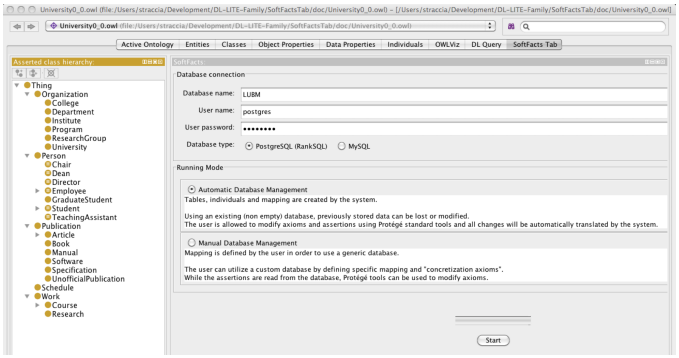


Fig. 7. Intialisation with ADM.

IV. SOFTFACTS PROTÉGÉ PLUG-IN

SoftFacts provides an graphical interface within the well-known Semantic Web ontology editor Protégé (see Fig. 2).

First of all, we launch the Protégé editor and may load, edit, and change an OWL 2 QL ontology. Then, we move to the SoftFacts Protégé plug-in, which starts with the connection to the underlying database engine. In the database connection panel (see Fig. 7), we specify the parameters to connect to the database. There are two initialisation modalities, that are described below.

A. Automatic Database Management (ADM)

In the Automatic Database Management (ADM) modality, the system builds automatically three database tables to store the assertions (the facts) of an OWL ontology, so that the user has not to care about the structure of the database. For

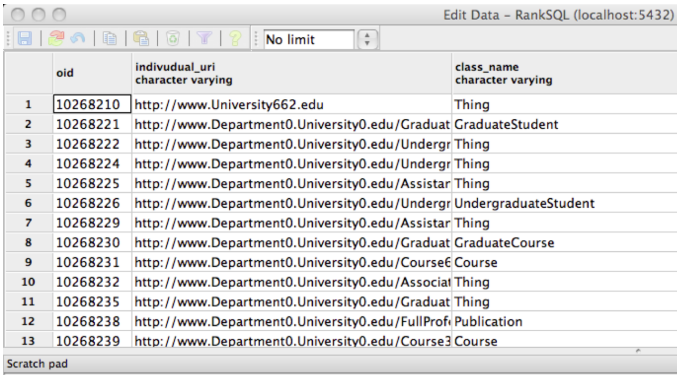


Fig. 8. The database table of concept instances.

instance, Fig. 7 displays the case of the LUBM ontology [6]. The SoftFacts Protégé plug-in starts then automatically to translate the LUBM ontology in SoftFacts syntax and creates and populates also three relational tables, with signature

class_instances(indivudual_uri[string], class_name[string])

object_properties_instances(individual_uri_1[string], property_name[string], individual_uri_2[string])

data_properties_instances(property_name[string], value_string[string], value_bool[string], value_real[float])

We recall that in OWL2 QL, one may assert that an individual a is instance of a concept A, a pair of individuals (a, b) is an instance of an object property T, and that a pair (a, v), where a is an individual and v is a value of some type, is an instance of a data type property D. So, for instance, a fact $Person(tom)$ is stored into the class_instances tables as tuple $\langle tom, Person\rangle$, a fact $hasFriend(tom, susan)$ is stored into the object_properties_instances table as the triple $\langle tom, hasFriend, susan\rangle$, while a fact $hasAge(tom, 32)$ is stored into the data_properties_instances table as the tuple $\langle tom, hasAge, -, -, 32\rangle$. As illustrative example, Fig. 8 shows an excerpt of the class_instances table.

In the SoftFacts Protégé plug-in, in the ADM modality, we may distinguish (see Fig. 2):

(i) In the upper panel, there are three tabs

- Axioms: here you see the SoftFacts axioms automatically generated from the ontology (expressions of the OWL ontology that cannot mapped into SoftFacts are dropped). This panel is not editable to avoid inconstancies with the ontology.
- Mapping: this panel shows the abstraction statements created to connect the abstract concepts of the ontology to the database. This panel is not editable to avoid inconstancies with the database.
- Macros: this panel shows some useful macros to be used at query time to perform top-k queries over the ontology. The panel is editable.

(ii) In the middle part of the main panel, we may define SoftFact queries to be submitted to the SoftFacts system, as

[6] \url{http://swat.cse.lehigh.edu/projects/lubm/University0_0.owl}.

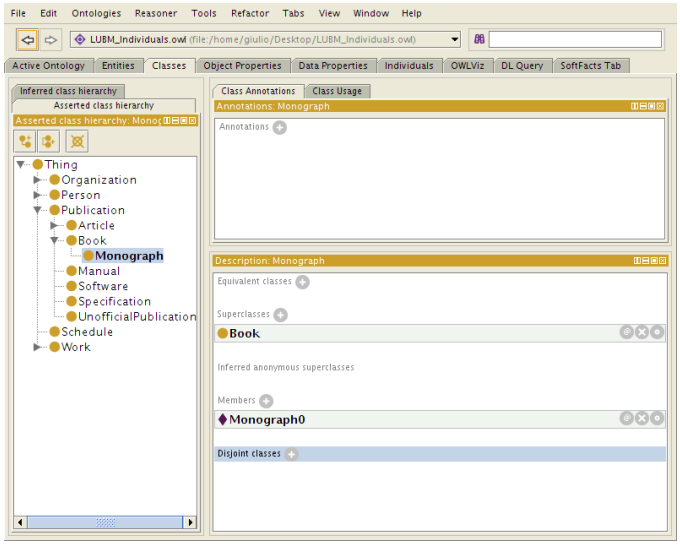


Fig. 9. Adding a new class and a new instance of it.

shown in Fig. 2.

(iii) In the lower part of the main panel, we may perform the following operations:

- We may safe the ontology (or parts of it) in SoftFacts syntax as file;
- We may synchronise modifications to the ontology (left panel) by means of the Protégé editor with the SoftFacts system (UpdateAll button).

1) Modifying the ontology: In the following, we show how to add a modification to the ontology and making it effective to the SoftFacts reasoner.

Suppose we would like to add the axiom $Monograph \sqsubseteq Book$ and to assert an individual Monograph0 as instance of the class $Monograph$ (see Fig. 9). To make these changes effective to SoftFacts, we use the UpdateAll button and after that we may notice the adding of the SoftFacts axiom in the axioms panel (Fig. 10).

B. Manual Database Management (MDM)

In the Manual Database Management (MDM) case, thought for an expert user, the user needs to set-up manually the interaction with the database in a consistent manner. Specifically, once an ontology has been loaded, the system does only translate the ontology axioms into SoftFacts specific axioms. It is then up to the user to define the mapping to the underlying database tables. The individuals in the ontology are not stored into the database. After the initialisation, the plugin is similar as for the ADM case, except that we have an additional editable *concretisation axioms* panel (and editable mapping panel). Here the user defines so-called *concretisation axioms* used to populate some specific concepts of the ontology. These axioms have on the right side abstract concepts, while have on the left side concrete concepts, *i.e.* concepts whose instances are stored into a database.

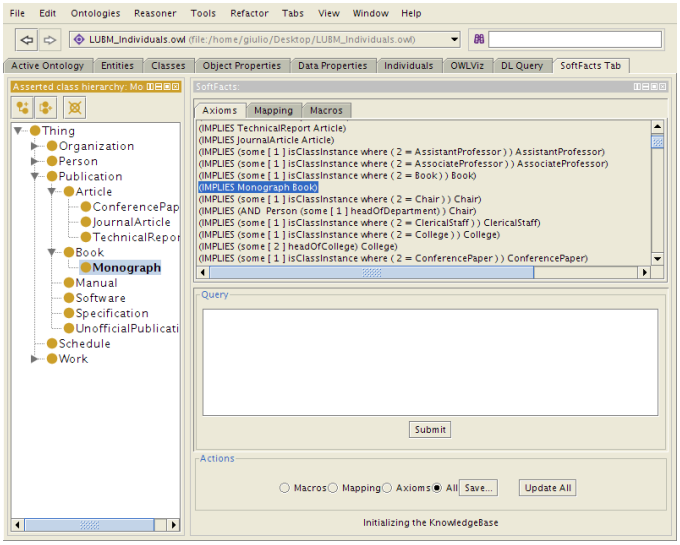


Fig. 10. SoftFacts Tab updated.

1) Concretisation axioms: For instance, assume that we have initialised the LUBM ontology according to MDM and let us assume that we consider the LUBM database that has been created for the ADM case, which has three tables. At first, we need to add the abstraction statements that related database tables to the concepts of the ontology:

$$\begin{aligned}
\mathsf{isClassInstance} &\mapsto \mathsf{class_instances(indivudual_uri[string],}\\
&\quad \mathsf{class_name[string])}\\
\mathsf{isDataPropertyInstance} &\mapsto \mathsf{data_properties_instances(property_name[string],}\\
&\quad \mathsf{value_string[string], value_bool[string],}\\
&\quad \mathsf{value_real[float])}\\
\mathsf{isObjectPropertyInstance} &\mapsto \mathsf{object_properties_instances(individual_uri_1[string],}\\
&\quad \mathsf{property_name[string], individual_uri_2[string])}
\end{aligned}$$

For instance, the first one says that isClassInstance is a binary relation that maps to the class_instances table of the LUBM database and whose columns are class_instances and class_name, respectively. These abstraction statements have to be added to the Mapping panel. Now, we may use concretisation axioms to populate some specific concepts starting from the database tables. Examples of concretisations axioms are the following:

$$\begin{aligned}
\mathsf{isClassInstance}[1].((\mathsf{[2]} = \mathsf{AdministrativeStaff})) &\sqsubseteq \mathsf{AdministrativeStaff}\\
\mathsf{isClassInstance}[1].((\mathsf{[2]} = \mathsf{Article})) &\sqsubseteq \mathsf{Article}\\
\mathsf{isObjectPropertyInstance}[1,3].((\mathsf{[2]} = \mathsf{advisor})) &\sqsubseteq \mathsf{advisor}[1,2]\\
\mathsf{isObjectPropertyInstance}[1,3].((\mathsf{[2]} = \mathsf{affiliateOf})) &\sqsubseteq \mathsf{affiliateOf}[1,2]\\
\mathsf{isDataPropertyInstance}[1,3].((\mathsf{[2]} = \mathsf{age})) &\sqsubseteq \mathsf{age}[1,2]
\end{aligned}$$

For the sake of explanation, the first one says that instances of the concept AdministrativeStaff are the projection on the first column of the relation isClassInstance where the second column's value is "AdministrativeStaff". Once all axioms and abstraction statements have been created, we need to push the UpdateAll button so that the SoftFacts systems becomes aware of all the changes.

2) Modifying the database: Now, we show how we need to update the SoftFacts systems to handle the situation in which we modify the database structure. So, suppose that now we would like to associate to the publication its publication year and let us assume that we modify the database with

```
CREATE TABLE publication_year ( publication_uri character varying NOT NULL,
                                year integer NOT NULL );
```

and populate the table via SQL, *e.g.*

```
INSERT INTO publication_year
VALUES ('http://www.Department0.University0.edu/FullProfessor3/Publication13', 2000);
```

Now we need to connect the new table to some new abstract relation using a SoftFacts abstraction statements. Hence, we define the abstract statement

$$\mathsf{publicationYear} \mapsto \mathsf{publication\_year(publication\_uri[STRING], year[INT])}$$

that creates a new abstract relation publicationYear that is related to the relational table publication_year. Additionally, we define an axiom that states that the domain of this relation are publications:

$$\mathsf{publicationYear}[1] \sqsubseteq \mathsf{Publication}$$

Finally, we push the UpdateAll button to update the SoftFacts system.

3) Top-k retrieval query: We are going now to use this additional data to submit a top-k query. Suppose we would like to have a ranked query, which is as follows: Find the top-10 ranked authors whose publications are scored according to the right-shoulder function (see Fig. 5) $rs(x; 2002, 2008)$ and that the final author's score is the sum of his publication scores. In our abstract syntax, this query may be written as

$$\begin{aligned}
q(x) \leftarrow\ & \mathsf{publicationAuthor}(x, y), \mathsf{publicationYear}(y, t),\\
& \mathsf{GroupedBy}(x),\\
& \mathsf{OrderBy}(s = \mathsf{SUM}[rs(t; 2002, 2008)]),\\
& \mathsf{Limit}(10)
\end{aligned}$$

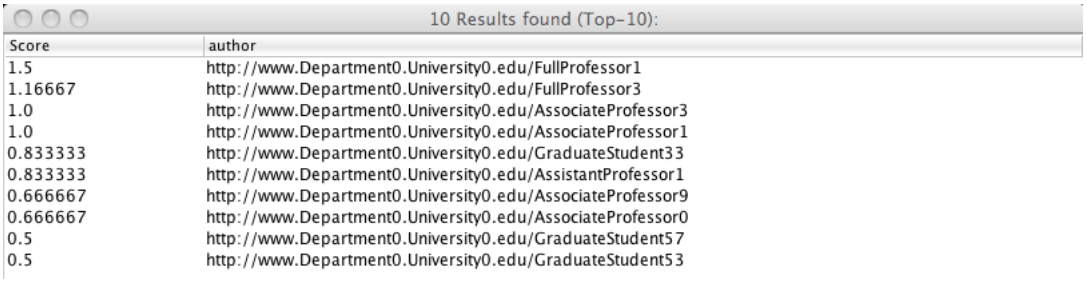
In concrete SoftFacts syntax, the query is

```
(RETRIEVE (?author) WHERE( (publicationAuthor ?pub ?author)
                           (publicationYear ?pub ?year) )
                           ORDERBY( SUM(&rs(2002 2008 ?year)) )
                           LIMIT (10) )
```

Submitting the query by mans of the Submit button we get the results as *e.g.* shown in Fig. 11.

V. Conclusion

The top-k retrieval problem is an important problem in logic-based languages for the Semantic Web. We have addressed this issue in the SoftFacts system, an ontology mediated top-k information retrieval system over relational databases. In SoftFacts, an ontology layer is used to define (in terms of a tractable DLR-Lite like description logic) the relevant abstract concepts and relations of the application domain, facts are stored into a relational database, accessed via an abstraction component. The results of a query may be ranked according to scoring functions. We have illustrate the architecture, the representation and the query language, sketched the reasoning algorithms of the SoftFacts system and its use within the Protégé editor. We refer the reader to [23] for technical details on the algorithms and experiments conducted with the SoftFacts system, which show good results from a scalability point of view (we tested with an ontology of more than 5000 axioms to represent CVs and 500000 CVs).



10 Results found (Top-10):

Score	author
1.5	http://www.Department0.University0.edu/FullProfessor1
1.16667	http://www.Department0.University0.edu/FullProfessor3
1.0	http://www.Department0.University0.edu/AssociateProfessor3
1.0	http://www.Department0.University0.edu/AssociateProfessor1
0.833333	http://www.Department0.University0.edu/GraduateStudent33
0.833333	http://www.Department0.University0.edu/AssistantProfessor1
0.666667	http://www.Department0.University0.edu/AssociateProfessor9
0.666667	http://www.Department0.University0.edu/AssociateProfessor0
0.5	http://www.Department0.University0.edu/GraduateStudent57
0.5	http://www.Department0.University0.edu/GraduateStudent53

Fig. 11. Top-10 retrieval.

REFERENCES

[1] F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05*, pages 364–369, Edinburgh, UK, 2005. Morgan-Kaufmann Publishers.

[2] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[3] Patrick. Bosc and Olivier Pivert. Sqlf: a relational database language for fuzzy querying. *IEEE Transactions on Fuzzy Systems*, 3(1):1–17, 1995.

[4] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data complexity of query answering in description logics. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR-06)*, pages 260–270, 2006.

[5] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Riccardo Rosati, and Marco Ruzzi. Data integration through DL-Lite$_a$ ontologies. In *Semantics in Data and Knowledge Bases, Third International Workshop, SDKB 2008, Nantes, France, March 29, 2008, Revised Selected Papers*, number 4925 in Lecture Notes in Computer Science, pages 26–47. Springer Verlag, 2008.

[6] Kevin Chen-Chuan Chang and Seung won Hwang. Minimal probing: Supporting expensive predicates for top-k queries. In *SIGMOD Conference*, pages 346–357, 2002.

[7] Ronald Fagin. Combining fuzzy information: an overview. *SIGMOD Rec.*, 31(2):109–118, 2002.

[8] Ian Horrocks, Lei Li, Daniele Turi, and Sean Bechhofer. The instance store: DL reasoning with large numbers of individuals. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, pages 31–40, 2004.

[9] Ihab F. Ilyas, Walid G. Aref, Ahmed K. Elmagarmid, Hicham G. Elmongui, Rahul Shah, and Jeffrey Scott Vitter. Adaptive rank-aware query optimization in relational databases. *ACM Transactions on Database Systems*, 31(4):1257–1304, 2006.

[10] Chengkai Li, Kevin Chen-Chuan Chang, and Ihab F. Ilyas. Supporting ad-hoc ranking aggregates. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD-06)*, pages 61–72, USA, 2006. ACM Press.

[11] Chengkai Li, Kevin Chen-Chuan Chang, Ihab F. Ilyas, and Sumin Song. RankSQL: query algebra and optimization for relational top-k queries. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (SIGMOD-05)*, pages 131–142, New York, NY, USA, 2005. ACM Press.

[12] Thomas Lukasiewicz and Umberto Straccia. Top-k retrieval in description logic programs under vagueness for the semantic web. In *Proceedings of the 1st International Conference on Scalable Uncertainty Management (SUM-07)*, number 4772 in Lecture Notes in Computer Science, pages 16–30. Springer Verlag, 2007.

[13] Jeff Z. Pan, Giorgos Stamou, Giorgos Stoilos, Edward Thomas, , and Stuart Taylor. Scalable querying service over fuzzy ontologies. International World Wide Web Conference (WWW 08), Beijing, 2008, 2008.

[14] Héctor Pérez-Urbina, Ian Horrocks, and Boris Motik. Efficient query answering for OWL 2. In *Proc. of the 8th International Semantic Web Conference (ISWC 2009)*, number 5823 in Lecture Notes in Computer Science, pages 489–504. Springer Verlag, 2009.

[15] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *Journal of Data Semantics*, 10:133–173, 2008.

[16] Azzurra Ragone, Umberto Straccia, Tommaso Di Noia, Eugenio Di Sciascio, and Francesco M. Donini. Vague knowledge bases for matchmaking in p2p e-marketplaces. In *4th European Semantic Web Conference (ESWC-07)*, number 4519 in Lecture Notes in Computer Science, pages 414–428. Springer Verlag, 2007.

[17] Azzurra Ragone, Umberto Straccia, Tommaso Di Noia, Eugenio Di Sciascio, and Francesco M. Donini. Fuzzy matchmaking in e-marketplaces of peer entities using Datalog. *Fuzzy Sets and Systems*, 160(2):251–268, 2009.

[18] Umberto Straccia. Answering vague queries in fuzzy DL-Lite. In *Proceedings of the 11th International Conference on Information Processing and Managment of Uncertainty in Knowledge-Based Systems, (IPMU-06)*, pages 2238–2245. E.D.K., Paris, 2006.

[19] Umberto Straccia. Towards top-k query answering in deductive databases. In *Proceedings of the 2006 IEEE International Conference on Systems, Man and Cybernetics (SMC-06)*, pages 4873–4879. IEEE, 2006.

[20] Umberto Straccia. Towards top-k query answering in description logics: the case of DL-Lite. In *Proceedings of the 10th European Conference on Logics in Artificial Intelligence (JELIA-06)*, number 4160 in Lecture Notes in Computer Science, pages 439–451, Liverpool, UK, 2006. Springer Verlag.

[21] Umberto Straccia. Towards vague query answering in logic programming for logic-based information retrieval. In *World Congress of the International Fuzzy Systems Association (IFSA-07)*, number 4529 in Lecture Notes in Computer Science, pages 125–134, Cancun, Mexico, 2007. Springer Verlag.

[22] Umberto Straccia. Managing uncertainty and vagueness in description logics, logic programs and description logic programs. In *Reasoning Web, 4th International Summer School, Tutorial Lectures*, number 5224 in Lecture Notes in Computer Science, pages 54–103. Springer Verlag, 2008.

[23] Umberto Straccia. Softfacts: a top-k retrieval engine for a tractable description logic accessing relational databases. Technical report, 2009.

[24] M. Vardi. The complexity of relational query languages. In *Proc. of the 14th ACM SIGACT Sym. on Theory of Computing (STOC-82)*, pages 137–146, 1982.

[25] Peter Vojtás. Fuzzy logic aggregation for semantic web search for the best (top-k) answer. In Elie Sanchez, editor, *Fuzzy Logic and the Semantic Web*, Capturing Intelligence, chapter 17, pages 341–359. Elsevier, 2006.

[26] Maria Zemankova and Abraham Kandel. Implementing imprecision in information systems. *Inf. Sci.*, 37(1-3):107–141, 1985.