SPARQL Extensions with Preferences: a Survey
Olivier Pivert, Olfa Slama, Virginie Thion

To cite this version:
Olivier Pivert, Olfa Slama, Virginie Thion. SPARQL Extensions with Preferences: a Survey. ACM Symposium on Applied Computing, Apr 2016, Pisa, Italy, France. 10.1145/2851613.2851690 . hal-01235190

HAL Id: hal-01235190
https://inria.hal.science/hal-01235190
Submitted on 23 Nov 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License
SPARQL Extensions with Preferences: a Survey

Olivier Pivert
Irissa-Enssat
University of Rennes 1
Lannion – France
pivert@enssat.fr

Olfa Slama
Irissa-Enssat
University of Rennes 1
Lannion – France
olfa.slama@irisa.fr

Virginie Thion
Irissa-Enssat
University of Rennes 1
Lannion – France
virginie.thion@irisa.fr

ABSTRACT

The last decade has witnessed an increasing interest in expressing preferences inside database queries. Even though most of the work has been devoted to relational databases, many proposals have also been made in the Semantic Web area in order to query RDF databases in a flexible way. This paper presents a survey of these approaches, classifies them, and points out research perspectives.

Categories and Subject Descriptors

H.2.3 [Languages]: Query Languages; H.3.3 [Information Search and Retrieval]: Query Formulation

Keywords

SPARQL; Preference Queries; RDF; Database

1. INTRODUCTION

In the last decade, database research has witnessed much interest in the W3C’s Resource Description Framework (RDF), which is considered to be the most appropriate knowledge representation language for representing, describing and storing Semantic Web data and associated meta-data. This graph data model makes it possible to represent heterogeneous Web resources in a common and unified way, taking into consideration the semantic side of the information and the interconnectedness between entities. SPARQL is the standard query language for querying RDF data.

RDF data are usually composed of large heterogeneous data including various levels of quality e.g., over relevancy, trustworthiness, preciseness or timeliness of data (see [Zaveri et al., 2014]). It is then necessary to offer convenient query languages that improve usability of such data. A solution is to integrate user preferences into queries, which allows users to retrieve data in a more flexible way. Motivations for integrating user preferences into database queries are manifold [Hadjali et al., 2011]. First, it appears to be desirable to offer more expressive query languages that can be more faithful to what a user intends to say. Second, the introduction of preferences in queries provides a basis for rank-ordering the retrieved items, which is especially valuable in case of large sets of items satisfying a query. Third, a classical query may also have an empty set of answers, while a relaxed (and thus less restrictive) version of the query might be matched by some items.

Introducing user preferences in queries has been a research topic for already quite a long time in the context of the relational database model. In the literature, one may find many flexible approaches suited to the relational data model: top-k queries [Bruno et al., 2002], the winnow [Chomicki, 2002] and Best [Torlone and Ciaccia, 2002] operators, skyline queries [Borzsony et al., 2001], Preference SQL [Kießling, 2002], as well as approaches based on fuzzy set theory [Tahani, 1977] [Bosc and Pivert, 1995] [Pivert and Bosc, 2012]. The literature about preference queries to RDF databases is not as abundant since this issue has started to attract attention only recently. In this paper, we present an overview of approaches that have been proposed to make SPARQL querying of RDF data more flexible, followed by a classification of these approaches.

The paper is organized as follows. Sections 2 and 3 briefly introduce background notions, namely the RDF data model and the SPARQL language. In Section 4, we present the approaches of the literature aiming at extending the SPARQL language with preference queries. We distinguish quantitative approaches from qualitative ones. These approaches and some open research perspectives are then discussed in Section 5.

2. RDF DATA

The Resource Description Framework (RDF) [W3C, 2014] uses a set of resource names, a set of literals and a set of blank nodes (i.e., unknown or anonymous resources) respectively denoted by $\text{URI}$, $L$ and $B$ in the following.

Let us consider a movie as a resource of the Web. A characteristic may be attached to the movie, like a director, a title, an actor or a genre. In order to express such a characteristic, the RDF data model uses a statement of the form of a triple $(s, p, o) \in (\text{URI} \cup B) \times \text{URI} \times (\text{URI} \cup L \cup B)$. The subject $s$ denotes the resource being described, the predicate $p$ denotes the property of the resource and the object $o$ denotes the property value. A triple states that the subject $s$ has a property $p$ with a value $o$.

For instance, the triple $(\text{Ocean’s Twelve}, \text{movie:actor}, \text{George Clooney})$ states that Ocean’s Twelve has George Clooney as an actor property, which can be interpreted as:
George Clooney plays in Ocean’s Twelve.

A set of RDF triples can be modeled by a directed labeled graph (called RDF graph or simply graph in the following) where for each triple \((s,p,o)\), the subject \(s\) and the object \(o\) are the nodes, and the predicate \(p\) corresponds to the edge from the subject to the object. RDF is then a graph-structured data model that maintains the basic notions of graph theory (such as node, edge, path, neighborhood, connectivity, distance, in-degree, out-degree, etc.).

Example 1. Let us take the following example extracted from the Internet Movie Database (IMDb)\(^1\): the resource http://data.linkedmdb.org/resource/film/97605 is a film, labeled and titled *The American*. It was released on 2010-09-01, written by the resource http://data.linkedmdb.org/resource/writer/741, named Rowan Joffe, directed by the resource http://data.linkedmdb.org/resource/director/9742, named Anton Corbijn and featured the resource http://data.linkedmdb.org/resource/actor/30516 named George Clooney, which is also an actor of the film http://data.linkedmdb.org/resource/film/5541 released on 2007-05-24. Figure 1 is a graphical representation of such data. ⋄

![Figure 1: Sample RDF graph extracted from IMDb](http://www.imdb.com/)

RDF provides a *schema definition* language called RDF Schema (RDFS), which allows to specify semantic deductive constraints on objects, subjects and properties of an RDF graph. It permits to declare objects and subjects as *instances* of given classes, and *inclusion statements* between classes and properties. It is also possible to relate the *domain* and *range* of a property to classes. RDF also declares entailment rules that allow to derive new triples from the explicit triples appearing in an RDF graph. Such implicit triples are part of the RDF graph even if they do not explicitly appear in it. They can be explicitly added to the graph. When all implicit triples are made explicit in the graph then the graph is said to be *saturated*. In the following, we consider that the RDF graph is saturated, as this is implicitly the case in all the approaches presented in the survey.

3. SPARQL

SPARQL [Prud and Seaborne, 2008] is the standard query language promoted by the W3C for querying RDF Data. It is a declarative query language based on graph pattern matching, in the sense that the query processor searches for sets of triples in the data graph that satisfy a pattern (i.e., set of triples containing variables) expressed in the query.

A SPARQL query has the general form given in Query 1, where the optional clause *PREFIX* is for abbreviating URIs, the clause *SELECT* is for specifying which variables should be returned, the clause *FROM* defines the datasets to be queried, and the clause *WHERE* contains the triple of the researched pattern.

```
PREFIX .. #Prefix declarations
SELECT .. #Result
FROM .. #Dataset definition
WHERE .. #Pattern
ORDER BY .., DISTINCT .., .. #Modifiers
```

**Query 1: Skeleton of a SPARQL query**

SPARQL also provides solution modifiers, which make it possible to modify the result set by applying classical operators like *ORDER BY*, *DISTINCT*, *LIMIT*, *PROJECTION*, or *OFFSET*.

Example 2. Query 2 is a simple SPARQL query taken from the IMDb database that aims to retrieve the names of the movies of the genre *Drama* featuring George Clooney, sorted in ascending order of their release date, and limited to 10 responses. The variables are the terms prefixed by the question mark symbol. ⋄

```
PREFIX dc: <http://purl.org/dc/terms/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX movie: <http://data.linkedmdb.org/resource/movie/>
SELECT ?Name WHERE {
  ?Actor movie:actor_name "George Clooney".
  ?Movie dc:date ?Date.
  ORDER BY ASC(?Date)
  LIMIT 10}
```

**Query 2: Simple SPARQL query**

In the following, we present the contributions that extend SPARQL by the integration of user preferences in queries.

4. PREFERENCE QUERIES ON RDF DATA

In the Semantic Web domain, some authors called for integrating the notion of preference into the SPARQL language in order to express requirements that faithfully reflect the needs of the users and produce discriminated answers. The proposed approaches may be classified into two categories according to their qualitative or quantitative nature. In the latter, preferences are expressed quantitatively by a monotone scoring function (the overall score is positively correlated with partial scores). In the qualitative category of approaches, preferences are defined through binary preference relations. Typical representatives of this category are approaches based on Pareto order, aimed at computing non-dominated answers (whose set constitutes a so-called skyline). Note that these approaches only yield a partial order, contrary to the quantitative ones.

\(^1\)http://www.imdb.com/
In the following, we first present quantitative approaches (Subsection 4.1), then qualitative ones (Subsection 4.2).

4.1 Quantitative approaches

All of the quantitative approaches presented hereafter share the same general principles: they are implicit and each involved preference is defined via an atomic scoring function allowing a score (aka. satisfaction degree) to be associated with each answer, making it possible to get a total ordering of the answers.

4.1.1 Fuzzy Extensions of SPARQL

SPARQL supports only a few standard ways of retrieval, all based on Boolean logic. Therefore, it is not possible to handle fuzzy conditions in user queries. In order to meet user needs more effectively, Cheng et al. [Cheng et al., 2010] propose a syntactical fuzzy extension of SPARQL called f-SPARQL. Their extension supports the expression of fuzzy conditions including (possibly compound) fuzzy terms, e.g., recent or young, and fuzzy operators, e.g., close to or at least, interpreted in a gradual manner. Most fuzzy predicates are assumed to be represented by a trapezoidal membership function (see for instance a possible representation of recent in Figure 2). Membership functions of the fuzzy predicates at least \( Y \) and close to \( Y \) are proposed in [Cheng et al., 2010].

![Membership function of recent](image)

**Figure 2:** Membership function of recent

The f-SPARQL extension of SPARQL concerns the \texttt{filter} clause whose syntax becomes \texttt{filter (\{X \theta FT\}) [with \( \alpha \)]}, where \( FT \) denotes a fuzzy term and \( \theta \) denotes an operator or comparator which can be fuzzy (such as close to (around), at least, and at most) or not (such as \( > \), \( < \), \( = \), \( \neq \), \( \sim \), \( \approx \), \( ! \approx \), between and not between). The optional parameter [with \( \alpha \)] specifies the smallest acceptable membership degree in the interval \([0, 1]\).

**Example 3.** The fuzzy query "Retrieve recent movies with George Clooney" is formulated by the f-SPARQL Query 3. If the IMDb RDF database is queried, then the movie entitled The American (cf. Figure 1) belongs to the answer, with a satisfaction degree of 0.25, which corresponds to the degree of membership of value 2010 to the fuzzy term recent (see Figure 2). The other movie from Figure 1, released in 2007, does not belong to the answer as it is not recent according to Figure 2.


**Query 3: An f-SPARQL query**

Now let us assume that the database of the running example embeds a rating value for each movie, through a property named \texttt{dc:rate} connecting a movie (URI resource) to a rating value (a label). When a user wants to express preferences on several attributes (e.g., date, rating, etc.), he/she may assign an importance to every partial preference. If no importance is specified, it is implicitly assumed that the partial degrees are aggregated by means of the triangular norm \( \text{min} \) which is commonly used in fuzzy logic to interpret the conjunction. In [Cheng et al., 2010], the authors propose to use a weighted mean in order to combine the partial scores coming from the different atomic preference criteria: \( \text{score}(A) = \sum_{i=1}^{n} \mu(A_i) \times w(F_i) \), where \( F = (F_1, ..., F_n) \) is the set of \texttt{filter} conditions, \( A_i \) is the property concerned by \( F_i \) in the candidate answer \( A, \mu(A_i) \) denotes the membership degree of the answer for \( F_i \), and \( w(F_i) \) denotes the weight assigned to \( F_i \), assuming \( \sum w(F_i) = 1 \).

**Example 4.** Consider the query "retrieve the recent (importance 0.2) movies featuring George Clooney with a high rating (importance 0.8)". It is expressed in f-SPARQL by Query 4.

**SELECT ?Movie ?Actor ?Date ?Rating WHERE { ?Actor movie:actor-name "George Clooney". ?Movie movie:actor ?Actor. \texttt{filter (\{?Date = recent\}) with 0.8.} \texttt{filter (\{?Rating = high\}) with 0.2.}**

**Query 4: f-SPARQL query with weights**

It is also possible to apply a threshold \( \alpha_i \) to an atomic fuzzy condition \( F_i \) (this threshold is associated with the underlying attribute in the \texttt{select} clause). Then, an answer is qualified only if its membership degree relatively to \( F_i \) is at least equal to \( \alpha_i \). Surprisingly, it does not seem that f-SPARQL makes it possible to specify a threshold on the global satisfaction degree. Let us notice that f-SPARQL queries may also involve a quantitative threshold \( k \), and then, only the top-\( k \) answers are returned.

The authors of [Cheng et al., 2010] exhibit a set of translation rules for three types of fuzzy terms (simple atomic terms, e.g., \texttt{recent}, modified fuzzy terms, e.g., \texttt{very recent}, and compound fuzzy terms, e.g., \texttt{popular and very recent}) and fuzzy operators. These translation rules are used to convert f-SPARQL queries into Boolean ones which are already supported by the existing implementations of standard SPARQL. The same principle was initially proposed in [Bosc and Pivert, 2000] in the context of relational databases to process SQLf (fuzzy) queries.

Some of the authors of [Cheng et al., 2010] proposed a variant of f-SPARQL called fp-SPARQL [Wang et al., 2012] that involves (i) an alternative way of interpreting modified fuzzy terms (i.e., an atomic fuzzy term modified by an adverb such as extremely, very, rather and so on), and (ii) an alternative way of interpreting compound fuzzy conditions where atomic predicates are assigned a priority.

4.1.2 Top-k Queries

Top-\( k \) queries [Bruno et al., 2002] are a popular class of queries that return only the \( k \) most relevant (best) tuples according to user’s preferences. The attribute values of each tuple are associated with a value or score using a simple
linear function. Top-\(k\)-queries have raised a growing interest in the last few years in the Semantic Web community [Bozzon et al., 2012, Magliacane et al., 2012, Dividino et al., 2012]. A major challenge is to make the processing of such queries efficient in a SPARQL-like setting. Classical top-\(k\)-SPARQL queries can be expressed by solution modifiers such as ORDER BY and LIMIT clauses that respectively order the result set and limit the number of results.

Example 5. Top-\(k\)-SPARQL Query 5 aims to find the best five offers of movies ordered by a function of user ratings and offer date where the \(g_i\)'s are scoring functions.

```
SELECT ?Movie ?Offer (g_1(?avgRating) + g_2(?date1) AS ?score) WHERE {
?Offer type:Movie ?Movie
?Movie hasAvgRating ?avgRating.
?Movie hasName ?Name.
?Movie hasDate ?date1.
}
ORDER BY DESC(?score) LIMIT 10
```

Query 5: Standard top-\(k\)-SPARQL-query

A naive processing of these queries relies on a materialize-then-sort procedure which entails an evaluation of all the candidate answers (i.e., those satisfying the condition in the WHERE clause), followed by a computation of the ranking function for each of them, even if only a small number (typically, \(k = 10\)) of answers is requested. Recent works have proposed solutions to optimize the evaluation of such queries. For instance, the authors of [Bozzon et al., 2012, Magliacane et al., 2012] introduce rank-aware operators as well as algebraic equivalences and laws involving these operators (like pushing an atomic preference over binary operators such as join, union, difference). The objective here is to derive an optimized query execution plan.

In [Dividino et al., 2012], the authors introduce an approach for top-\(k\) querying RDF data annotated with provenance information. In this context, annotations may concern the origin, history, truthfulness, or validity of an RDF state-

4.2 Qualitative approaches

In the relational database domain, qualitative approaches to preference queries have attracted a large interest, in particular skyline queries [Borzsony et al., 2001], which aim to filter an \(n\)-dimensional dataset \(S\) according to a set of user preference relations and return only the tuples of \(S\) that are not dominated in the sense of Pareto order.

Let us consider two tuples \(t = (u_1, \ldots, u_n)\) and \(t' = (u'_1, \ldots, u'_n)\) from \(S\) (reduced to the attributes on which a preference is expressed). Tuple \(t\) dominates (in the sense of Pareto order) tuple \(t'\), denoted by \(t \succ t'\), iff \(t\) is at least as good as \(t'\) in all dimensions and strictly better than \(t'\) in at least one dimension. This may be represented by:

\[
t \succ t' \iff \forall i \in \{1, \ldots, n\}, t.u_i \succeq t'.u'_i \text{ and } \exists j \in \{1, \ldots, n\} \text{ such that } t.u_j \succ_j t'.u'_j
\]

Example 6. Let us assume that a user is looking for a good movie to watch, preferring a movie which is recent and high rated. Consider three movies \(M_1\) (date 2015, rating 5.8), \(M_2\) (date 2013, rating 4), and \(M_3\) (date 2014, rating 8). Movie \(M_1\) is more recent and has a higher rating than \(M_2\). So, \(M_1\) dominates \(M_2\). Nevertheless, \(M_1\) does not dominate \(M_3\) since \(M_1\) is more recent than \(M_3\) but has a worse rating than \(M_3\). Hence, the skyline result is \(\{M_1, M_3\}\).

In the literature, some works [Siberski et al., 2006, Guerreoussova et al., 2013] have dealt with the expression and evaluation of skyline queries in a SPARQL-like language.

In [Siberski et al., 2006], SPARQL is extended with a PREFERRING clause in order to support the expression of multidimensional user preferences. This extension is based on the principle underlying skyline queries, i.e., it aims to find the nondominated tuples. Syntactically, the extension consists in extending SPARQL with a new clause PREFERRING that defines preferences. Two types of preferences may be included: Boolean preferences (where the answers that meet the condition are favored over those which do not) and scoring preferences (introduced by the keywords HIGHEST or LOWEST, where the elements with a higher value are favored over those with a lower value and vice versa).

Example 7. Let us consider that a user has the following preferences: \((P_1)\) prefer the movies rated “excellent” to the “very good” ones (Boolean preference), \((P_2)\) prefer the movies whose projection time is between 3pm and 11pm (Boolean preference), and \((P_3)\) prefer the movies projected the latest (scoring preference) provided that they are projected between 3pm and 11pm.

In the absence of a skyline functionality, one would use the classical SPARQL Query 6 that returns the movies satisfying the Boolean conditions, ordered according to the time when the projection starts.

```
SELECT ?Movie ?Title WHERE {
?Movie dc:has-rating ?rating.
FILTER (?rating = ft:very-good || ?rating = ft:excellent) }
ORDER BY DESC(?ProjectionStarts >= 3pm || ?ProjectionEnds <= 11pm)
```

Query 6: Query in SPARQL (ordered answer)

Note that Query 6 returns dominated movies, but only at the bottom of the list of the answers. In the extended SPARQL version of [Siberski et al., 2006], lines 8 to 11 of Query 6 are replaced by:

```
8 PREFERRING
9 ?rating = ft:excellent
10 AND
11 (?ProjectionStarts >= 3pm || ?ProjectionEnds <= 11pm)
12 CASCADE HIGHEST(?ProjectionStarts)
```

Query 7: Skyline extension of SPARQL [Siberski et al., 2006]

Lines 1 to 7 represent the graph patterns and hard constraints. Line 9 corresponds to preference \(P_1\), lines 11 and 12 correspond to \(P_2\), and line 13 corresponds to \(P_3\). The CASCADE clause in line 13 specifies that \(P_1\) is evaluated if and only if two answers are equivalent with respect to \(P_2\)
The authors give the semantics of the new constructs aimed to compute a skyline with SPARQL, but they do not deal with query processing/optimization aspects.

The approach proposed by Gueroussova et al. in [Gueroussova et al., 2013] is based on [Siberski et al., 2006] but the authors i) introduce user preferences in the FILTER clause, ii) replace the CASCADE clause by a PRIOR to clause in the spirit of Preference SQL [Kießling et al., 2011], iii) introduce new comparators that may be used to specify atomic preferences: BETWEEN, AROUND, MORE THAN, and LESS THAN. This extension of SPARQL is called PrefSPARQL. It supports, not only the expression of qualitative preferences (skyline) but also of conditional ones (if-then-else). A PrefSPARQL query returns a set of partially ordered tuples according to the satisfaction of the preferences.

Example 8. In order to illustrate the form taken by skyline queries in PrefSPARQL, we consider the case of a user who prefers movies rated “excellent”, a projection time in the evening (between 6pm and 11pm), and prefers later projections to earlier ones provided that they take place in the evening. Query 8 expresses this in PrefSPARQL.

```
SELECT ?Movie ?Title WHERE {
    ?Movie dc:starts ?ProjS.
    ?Movie dc:ends ?ProjE.
    ?Movie dc:has-rating ?rating.
    PREFERING ?rating = ft:excellent AND
        (?ProjS BETWEEN (6pm, 11pm) AND
         ?ProjE BETWEEN (6pm, 11pm)
        PRIOR TO HIGHEST (?ProjE))
}
```

Query 8: Skyline query in PrefSPARQL

Example 9. To illustrate conditional preferences, let us now assume that a user prefers movies that start after 7:30pm on the weekdays and before 7pm during the weekends, as formulated in Query 9.

```
SELECT ?Movie WHERE {
    ?Movie dc:day ?D; dc:starts ?ProjS.
    PREFERING
        (IF (?D = ‘Saturday’ || ?D = ‘Sunday’) THEN ?ProjS < 7pm ELSE ?ProjS >= 7:30pm)
}
```

Query 9: Conditional preference in PrefSPARQL

The authors of [Gueroussova et al., 2013] show that PrefSPARQL preference queries can be expressed in SPARQL 1.0 and SPARQL 1.1 using OPTIONAL queries or features available in SPARQL 1.1 such as NOT EXISTS.

Finally, let us also mention [Chen et al., 2011], which presents an efficient approach for processing skyline queries in an RDF data context, using a vertically partitioned schema model which is a common model to store such data.

5. DISCUSSION

In the previous section, we have presented a survey of approaches from the literature that aim to extend the SPARQL language with preferences. We now further discuss these approaches and outline a few open research perspectives.

Our first observation concerns the limited expressiveness of the approaches. Indeed, all of them are straightforward adaptations of proposals made in the relational database context: they make it possible to express preferences on the values of the nodes, but not on the structure of the RDF graph (structural preferences may concern the strength of a path, the centrality of nodes, etc.). A still open research perspective is thus to define a flexible extension of SPARQL including constructs from preference query languages proposed in a graph database context, see e.g., [Pivert et al., 2014], taking as a basis, one of the (nonflexible) extensions of SPARQL involving navigational functionalities, e.g., [Pérez et al., 2010], [Alkhateeb et al., 2009], etc.

There is also a real need for a flexible SPARQL that takes into account RDF graphs where data is described by intrinsic weighted values, attached to edges or nodes. This weight may denote any gradual notion like a cost, a truth value, an intensity or a membership degree. For instance, in the real world, the information stored on the Web, as well as its metadata are far from being perfect and are represented by vague/imprecise knowledge. An imprecise information may be of the form “A movie is recent with the degree of truth 0.9”. The RDF data model should be enriched in order to represent such information, and new query languages should be defined. A first step in this direction is the approach proposed by Buche et al. [Buche et al., 2008] that takes into account RDF graphs containing fuzzy annotations. Buche et al. then define a flexible querying system, which consists in translating fuzzy RDF annotations into fuzzy conceptual graphs and using a so-called approximate-projection operation in order to compare query in the form of a conceptual graphs with fuzzy annotation graphs. Concerning RDF extensions, Cedeño and Candan [Cedeño and Candan, 2011] propose an extension of the RDF model embedding weighted edges and an extension of SPARQL to support this feature, allowing new path predicates to express nodes reachability and the ability to express ranked queries. This approach takes the weights into account in order to rank the answers, but does not propose any means to express preferences in user queries. To our knowledge, none of the existing approaches aims to define a general purpose flexible version of SPARQL to weighted RDF databases, which remains an open research perspective.

Another related research perspective is SPARQL extensions for data-quality-aware preference queries. As far as we know, this research problem has not been tackled yet except for [Hartig, 2009], which only considers the trustworthiness dimension.

Table 1 summarizes the main features of the approaches presented in the survey, as well as the few others briefly tackled in the discussion.

Acknowledgment

This work has been partially funded by the French DGE (Direction Générale des Entreprises) under the project ODIN (Open Data INtelligence).

6. REFERENCES


<table>
<thead>
<tr>
<th>Approach</th>
<th>SPARQL extension</th>
<th>Preference queries</th>
<th>Type</th>
<th>Ranking</th>
<th>Weighted graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Cheng et al., 2010]</td>
<td>yes</td>
<td>yes</td>
<td>Fuzzy-set-based</td>
<td>Total order</td>
<td>no</td>
</tr>
<tr>
<td>[Wang et al., 2012]</td>
<td>yes</td>
<td>yes</td>
<td>Fuzzy-set-based</td>
<td>Total order</td>
<td>no</td>
</tr>
<tr>
<td>[Bozzon et al., 2012][Magliacane et al., 2012]</td>
<td>yes</td>
<td>yes</td>
<td>Top-k-queries</td>
<td>Total order</td>
<td>no</td>
</tr>
<tr>
<td>[Dividino et al., 2012]</td>
<td>yes</td>
<td>yes</td>
<td>Top-k-queries</td>
<td>Total order</td>
<td>no</td>
</tr>
<tr>
<td>[Siberski et al., 2006]</td>
<td>yes</td>
<td>yes</td>
<td>Skyline</td>
<td>Partial order</td>
<td>no</td>
</tr>
<tr>
<td>[Gueroussova et al., 2013]</td>
<td>yes</td>
<td>yes</td>
<td>Skyline</td>
<td>Partial order</td>
<td>no</td>
</tr>
<tr>
<td>[Chen et al., 2011]</td>
<td>yes</td>
<td>yes</td>
<td>Skyline</td>
<td>Partial order</td>
<td>no</td>
</tr>
<tr>
<td>[Buche et al., 2008]</td>
<td>no</td>
<td>yes</td>
<td>Fuzzy-set-based</td>
<td>Total order</td>
<td>Fuzzy annotations</td>
</tr>
<tr>
<td>[Cedeño and Candan, 2011]</td>
<td>yes</td>
<td>no</td>
<td>Top-k-queries</td>
<td>Partial order</td>
<td>Weighted edges</td>
</tr>
</tbody>
</table>

Table 1: Main features of the preference query approaches


