Semantic Web 13 (2022) 659–682
DOI 10.3233/SW-212895
IOS Press



# Handling qualitative preferences in SPARQL over virtual ontology-based data access

Marlene Goncalves [a,b,*], David Chaves-Fraga [b] and Oscar Corcho [b]

[a] *Computer Science and Information Technology, Universidad Simón Bolívar, Venezuela*
*E-mail: mgoncalves@usb.ve*
[b] *Ontology Engineering Group, Universidad Politécnica de Madrid, Spain*
*E-mails: marlene.gdasilva@upm.es, dchaves@fi.upm.es, ocorcho@fi.upm.es*

**Editors:** Axel-Cyrille Ngonga Ngomo, University of Paderborn, Germany; Muhammad Saleem, University of Leipzig, Germany; Ruben Verborgh, Ghent University – imec, Belgium
**Solicited reviews:** Aidan Hogan, Universidad de Chile, Chile; Ruben Taelman, Ghent University – imec, Belgium; one anonymous reviewer

**Abstract.** With the increase of data volume in heterogeneous datasets that are being published following Open Data initiatives, new operators are necessary to help users to find the subset of data that best satisfies their preference criteria. Quantitative approaches such as top-k queries may not be the most appropriate approaches as they require the user to assign weights that may not be known beforehand to a scoring function. Unlike the quantitative approach, under the qualitative approach, which includes the well-known skyline, preference criteria are more intuitive in certain cases and can be expressed more naturally. In this paper, we address the problem of evaluating SPARQL qualitative preference queries over an Ontology-Based Data Access (OBDA) approach, which provides uniform access over multiple and heterogeneous data sources. Our main contribution is Morph-Skyline++, a framework for processing SPARQL qualitative preferences by directly querying relational databases. Our framework implements a technique that translates SPARQL qualitative preference queries directly into queries that can be evaluated by a relational database management system. We evaluate our approach over different scenarios, reporting the effects of data distribution, data size, and query complexity on the performance of our proposed technique in comparison with state-of-the-art techniques. Obtained results suggest that the execution time can be reduced by up to two orders of magnitude in comparison to current techniques scaling up to larger datasets while identifying precisely the result set.

Keywords: Qualitative preference, Skyline, OBDA, query translation, R2RML

## 1. Introduction

Eliciting and exploiting preferences in query evaluation over relational databases and triple stores has attracted sustained interest in the last two decades [1,11–13,19,25,26,30,41,44,47]. Such interest is motivated by the need of users who are not database experts but are willing to explore large datasets, commonly coming from the integration of multiple and heterogeneous data sources [25,41,44,47]. Usually, these users do not know, a priori, what useful information they can extract from this data or they do not have a particular result in mind until it is discovered as an outcome of their data exploration process. During data exploration, they try to identify useful information according to their preferences by means of eliciting queries that best meet their criteria. Typical examples include travelers

*Corresponding author. E-mail: mgoncalves@usb.ve.

1570-0844 © 2022 – The authors. Published by IOS Press. This is an Open Access article distributed under the terms of the Creative Commons Attribution License (CC BY 4.0).

looking for the best deals on accommodation to visit a city or users looking for laboratories with the best offers on PCR tests in a range of hours more appropriate for them during the COVID-19 global pandemic. Database engines should be able to filter useful information to support the requirements of these non-expert users, i.e., evaluating queries that represent the desirable properties over the final result during the user's search process. This kind of user expects such queries to be easily posed, correctly interpreted by the engine, and computationally efficient.

Introducing preference criteria into queries has been tackled in the context of relational databases (RDB) [11–13,30–32]. They are often used to filter information and thus reduce the volume of data being displayed to the user. At the level of preference criteria, two different approaches can be pursued: qualitative and quantitative. In the quantitative approach [1,26], preference criteria are specified by means of scoring functions that assign a numerical score to each instance of the query response. Thus, an instance $A$ is preferred to an instance $B$ if the score of $A$ is higher than the score of $B$. The problem of lack of expressiveness of this approach is well known in utility theory [22] where preferences are only represented by numerical scoring functions which are not necessarily easy to define by any user, whereas preferences in the qualitative approach can be expressed more naturally. For example, a typical scoring function for finding cheap hotels near the sea is a weighted average of price and distance, but the user should know how to define weights. In addition, the user should normalize the values of price and distance because they have different units. In contrast, under the qualitative approach, the user may simply express his preferences as minimizing price and distance, which is more intuitive. The qualitative approach is strictly more general than the quantitative one since preference queries are based on the observation that expressions, such as "I prefer more A than B if C", are easily stated by users when asked about their preferences. For instance, when a customer wants to book an appointment for a PCR test, it is easy for him to say that he prefers closer laboratories with more economical prices and appointments available after 17:00 on weekdays. There are two reasons for expressing qualitative preference queries [19]. First, it is desirable to offer more expressive query languages that allow the user to express what he is really trying to specify. Second, a classic query may also return a set of empty answers, while a qualitative one may produce at least some answers. Under this scenario, it should be optimal if a query engine is able to directly optimize such expressions of preference in a query.

Although qualitative preference queries have been widely studied in relational databases (RDB) [1,19,26], the literature on them in the context of Semantic Web is not as abundant. There are multiple examples in the literature that can be interpreted as preference search [5,44]. Some extensions of the SPARQL query language to incorporate user qualitative preferences have been also studied [41,44,47], and even, Patel-Schneider and colleagues [41] extended SPARQL to correctly handle any preference relation in terms of computing the transitive closure of a preference relation over candidate solutions. A particular case of qualitative preference queries are the skyline ones [6] which have been widely studied in RDB [28]. Also, a set of client-server based algorithms have been introduced to evaluate skyline queries over knowledge graphs using standard query interfaces, such as SPARQL endpoints and TPF (Triple Pattern Fragments), with no control over how the knowledge graph is stored [29]. In our previous research [24], focused on skyline queries, we have already demonstrated that the use of physical implementations for the skyline operator can significantly improve the query performance w.r.t the evaluation of either queries that were translated to SPARQL using query rewriting techniques to comply with the SPARQL standard or the skyline operator on the top of RDF triplestore after evaluating the other operators such as joins, projection, etc. Although this means that qualitative preference queries can be executed over SPARQL endpoints, the lack of techniques that exploit the data storage structures in triplestores to specifically deal with the complexity of these queries can have a negative impact on their evaluation. Thus, the performance of these queries over RDF-based knowledge graphs is still low.

In addition, Virtual Knowledge Graphs (VKG) provide access to data in terms of an existing ontology in accordance with a set of mapping rules [42], either by generating materialized views (RDF files) [18,27] or by translating SPARQL queries into queries that are supported by the underlying source, which is known as virtualization [7,43]. The latter is specially relevant when a qualitative preference query has to be evaluated because: *i*) it ensures up-to-date results at the moment of execution; *ii*) the preferring clause can be pushed down to the underlying data management system (e.g., an RDBMS) exploiting the benefits of the proposed preference physical operators to improve query performance [31,32]; *iii*) to the best of our knowledge there are no works in the literature for evaluating SPARQL qualitative preference queries on VKG. In this article, we are interested in the evaluation of SPARQL qualitative preference queries over data that are not only available in RDF, as in the aforementioned works, but in relational databases.

*Problem statement and research objective* In this paper, we focus on the problem of evaluating SPARQL qualitative preference queries during the process of constructing a virtual knowledge graph. We are interested in determining the feasibility of such types of query evaluation, understanding the correctness and completeness of our approach, and then determining whether its performance is adequate in comparison with current qualitative preference query approaches over RDF.

*Approach* We describe Morph-Skyline++, a virtual knowledge graph approach for qualitative preference queries. Based on the SPARQL-to-SQL query translation approach defined by Chebotko and colleagues [10], Morph-Skyline++ translates and optimizes qualitative preference queries from SPARQL to SQL by means of a set of R2RML mappings. The approach includes a novel algorithm QualQT, which pushes down the evaluation of the preferring clause, delegating its treatment to a physical operator of the RDBMS.

*Evaluation* We use the Linked Movie Database (LinkedMDB) and adapt the benchmark for qualitative queries defined by Mandl and colleagues [38], with SPARQL queries. We compare our work against state-of-the-art tools. This evaluation enables us to understand the impact that queries of different complexities and dataset sizes have on a SPARQL-to-SQL qualitative preference query evaluation. The results of the experiments suggest that all these variables impact the total query execution time and approaches based on VKG can overcome native SPARQL methods up to two orders of magnitude. Additionally, as an initial study, we run two of the queries on gas stations from Patel-Schneider and colleagues' work [41] which were defined on an extension of a preference operator that correctly handles any preference relation. To the best of our knowledge, no benchmarks have been defined nor experimental studies conducted for these types of queries.

*Contributions* Our contributions can be summarized as follows: *i*) a formal definition of the problem of evaluating SPARQL qualitative preference queries on virtual knowledge graphs; *ii*) Morph-Skyline++, an approach based on the SPARQL-to-SQL translation approach proposed by Chebotko and colleagues [10] that is able to evaluate SPARQL qualitative preference queries over RDB; *iii*) the definition and implementation of QualtQT, an algorithm based on SPARQL-to-SQL translations taking advantage of specific physical operators for these queries; *iv*) an empirical evaluation of the behavior of Morph-Skyline++ over two benchmarks with queries of different complexities. *v*) a preliminary experimental study on the Patel-Schneider and colleagues' extension to SPARQL [41] for qualitative preferences, translating this extension back into SPARQL and using recursive queries to compute the transitive closure of a preference relation over candidate solutions. The specification of a syntax and semantics for the extension of SPARQL to handle preferences is based on PrefSPARQL grammar [25] although we include the DIFF directive which specifies grouping criteria in skyline queries. Finally, unlike our previous work [24] that only addresses the evaluation of SPARQL skyline queries on virtual knowledge graphs, in this work, we focus on qualitative preference queries which are more general than skyline ones.

The remainder of this article is structured as follows: Section 2 motivates the problem of evaluating qualitative preference queries on virtual knowledge graphs. Section 3 presents related work in qualitative preference queries and OBDA. Sections 4–5 describe our OBDA-based approach, the background, and the proposed solution which is implemented in Morph-Skyline++. Section 6 reports on and discusses the results of our empirical study, and finally, Section 7 concludes and gives insights for future work.

## 2. Motivating example

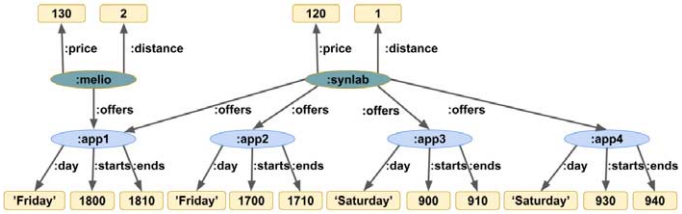
Suppose a customer will travel from Madrid to Germany and he has decided to get the RT-PCR test in Madrid on the Friday or Saturday before his trip. The customer is interested in finding the laboratories that meet the following two conditions in the query $Q_1$: *i*) are the cheapest, and *ii*) are the closest, and among those laboratories, he prefers the ones that have available earlier appointments *i*) before 10:00 if is Saturday, and *ii*) before 10:00 or after 17:00 if is Friday, and he also prefers earlier appointments ($\leqslant$ 10:00) over later ones ($\geqslant$ 17:00). All of these specifications are equally important for the customer. Following our customer's criteria, a laboratory will be taken into account if there is no other laboratory with better price, distance, and appointments available at the desired time. This set of laboratories will compose the preferred ones. Formally, the set of laboratories preferred by the user is composed of

```
SELECT *
WHERE {
      ?l :price ?price; :distance ?dist .
      ?l :offers ?a; :starts ?s; :ends ?e; :day ?day .
}
PREFERRING LOWEST ?price AND LOWEST ?dist AND
           (?s<=1000) PRIOR TO (?s>=1700) AND
           LOWEST IF (?day="Saturday")
                     THEN (?s<=1000)
                   ELSE IF  (?day="Friday")
                     THEN (?s>=1700 || ?s<=1000)
                   ELSE (false)
```

(a) A SPARQL Qualitative Preference Query Extended with a Preferring Clause

```
SELECT *
FROM laboratory l,offers o,appointment a
WHERE l.lab_id = o.lab_id AND o.app_id = a.app_id
PREFERRING LOWEST price AND LOWEST distance AND
           (starts<=1000) PRIOR TO (starts>=1700)
           AND LOWEST IF (day="Saturday")
                       THEN (starts<=1000)
                     ELSE IF (day="Friday")
                       THEN (starts>=1700 ||
                             starts<=1000)
                     ELSE (false)
```

(b) A SQL Qualitative Preference Query Extended with a Preferring Clause




(c) An RDF Graph

**laboratory**

| lab_id | price | distance |
|---|---|---|
| melio | 130 | 2 |
| synlab | 120 | 1 |

**offers**

| lab_id | app_id |
|---|---|
| melio | app1 |
| synlab | app1 |
| synlab | app2 |
| synlab | app3 |
| synlab | app4 |

**appointment**

| app_id | day | starts | ends |
|---|---|---|---|
| app1 | Friday | 1800 | 1810 |
| app2 | Friday | 1700 | 1710 |
| app3 | Saturday | 900 | 910 |
| app4 | Saturday | 930 | 940 |

(d) A Relational Database

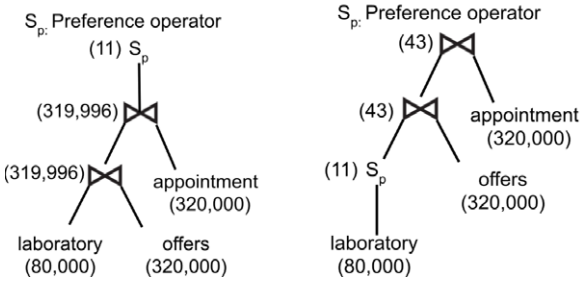
Fig. 1. Sample qualitative preference queries for booking a PCR appointment. The queries identify cheaper and closer laboratories with earlier appointments before 10:00 or after 17:00 on Friday or Saturday. Also, the SPARQL query is translated into SQL, and the RDB results are converted to RDF.

those that are not dominated by any other laboratory. Given $A$ and $B$ laboratories, $A$ dominates $B$, if $A$ has the same or better values in price, distance, and appointment schedule as $B$, and it has a better value in at least one of them. To illustrate qualitative preferences, consider the query $Q_1$ expressed in SPARQL as shown in the Fig. 1(a) and a knowledge graph that contains laboratory properties and their appointment offerings in Fig. 1(c).

Following these criteria, the appointments app2, app3, and app4 are the non-dominated ones, i.e., there is no other appointment with values better than them in :starts. Additionally, the appointment app2 dominates app1 because although its schedule is after 17:00, it is earlier than app1. In addition to this, synlab dominates melio because it has better price and distance, and equal value in the appointment.

The graph of the Fig. 1(c) can be seen as an RDF view defined over a relational database. If we consider that most PCR appointment booking systems are based on RDB technology and a change in their data model can be costly and time-consuming, the OBDA approach [42] can be a solution that ensures the results are always up to date, converting these data to RDF and making them available without renewing the entire software infrastructure. Under an OBDA approach where an RDF graph can then be queried using SPARQL, by translating the SPARQL queries into SQL queries over the relational database, the SPARQL query extended with a preferring clause from Fig. 1(a) can be translated according to R2RML mappings into a SQL query extended with a preferring clause as in Fig. 1(b), and after executing the SQL query extended with a preferring clause on data in Fig. 1(d), the results produced by the relational engine are subsequently transformed into a SPARQL resultset which is described through the concepts of the ontology and defined mappings.

Preference-based operators integrated as physical implementations inside relational database engines have shown substantial benefits [6,8,38]. Figures 2(a)–2(b) depict two query plans for the query $Q_1$ where the preference-based operator is at the plan root or is pushed down, respectively, and the number of instances is placed below the node and the number of intermediate results is placed to the left of the operator. Figure 2(c) reports the average execution time in seconds of the query $Q_1$ only with skyline preferences. It was executed 5 times using the state-of-the-art tools [29,47] over our motivating dataset transformed into RDF by means of SDM-RDFizer [27] and scaled-up to a 10.000 scale value by using VIG [35]. These state-of-the-art tools evaluate preferences on top of triplestores [29,47].




(a) The preference-based operator on top of the query plan

(b) The preference-based operator is pushed down into the query plan

| SPREFQL | TPF | brTPF | skyTPF |
|---|---|---|---|
| 55.45 | 138.51 | 32.42 | - |

(c) Average execution times (sec)

Fig. 2. Two query plans for a preference query. When the preference-based operator is integrated into the engine, query performance is improved. Intermediate results are shown over each step of the query plans using brackets.

It is important to highlight that skyTPF was not reported because it returned an empty answer. These state-of-the-art tools evaluate the preference-based operator on top of the database engine in a similar pattern to the plan shown in the Fig. 2(a). As well as the projection and selection operators that are pushed down in the query plan, a good heuristic is to apply a preference-based operator early because it reduces the size of the intermediate results by being an operator that filters data based on preferences. Pushing down the preference-based operator as in Fig. 2(b) would result in a lower execution time. Despite the fact that state-of-the-art tools have proposed optimizations to improve the performance of a preference query (e.g., Börzsönyi and colleagues [6] avoid either comparing each input instance against all input instances in the best of the cases), they ignore the engine capabilities underlying the data model, which results in higher execution time ($\geqslant$ 32.42). In this article, our objective is to apply a SPARQL-to-SQL translation where we take into account the capabilities of the underlying data model, exploiting them to improve the preference query performance

## 3. State of the art

There has been much interest in integrating the notion of preferences into query languages to express requirements that accurately reflect user needs [1,11–13,19,25,26,30,41,44,47]. The proposed approaches can be classified into two categories according to their quantitative or qualitative nature. In the former, preferences are expressed quantitatively through a scoring function to determine an ordering over query results. Closer to our work, in the latter, preferences are denoted by binary preference relationships yielding a partial order, contrary to the quantitative approach. Lacroix and Lavency [34], Chomicki and colleagues [11–13], Kießling and colleagues [30–32], and Börzsönyi and colleagues [6] defined foundational theories relating preferences to database systems and proposed extensions to SQL that support the specification of qualitative preference queries. These authors defined a logical formalism that allows the use of qualitative preference criteria within SQL, where qualitative preferences are represented by strict partial orders through an operator called winnow. The winnow operator is a generalization of the skyline operator [6]. A skyline operator returns a partially ordered set of instances whose order is induced by criteria composed of conditions on equally important parameters. For example, a typical skyline query is to find hotels that are both cheap and close to the beach.

The problem of efficiently computing the skyline has been widely studied in the literature [3,6,14,23]. Bentley and colleagues [4] proposed the first skyline algorithm, referred to as the maximum vector problem and based

on the divide & conquer principle. Progress continued to be made on how to compute efficiently such queries in a relational system and over large datasets, and thus, the skyline approach [6] was defined in the context of databases to distinguish the best tuples that satisfy a given ranking condition. Subsequently, several authors proposed algorithms that exploit data ordering properties or index structures to improve the skyline computation [3,14,20,23,33]. In the Semantic Web context, Keles and Hose [29] presented a set of server-client based algorithms to evaluate skyline queries over knowledge graphs using standard query interfaces for RDF. They are focused on the optimization over client architectures, so they assume no control over how the data is stored (e.g., indexes, internal structures, etc), hence, they cannot exploit them to optimize the performance and quality of the queries at the server side. However, in our context, we have control over the data stored by the database engine. In this work, preferences more general than skyline are not supported. In addition, these works have not focused on the challenges associated with implementing preferences as both a logical and physical operator within a engine and how the interaction of the preference operator with other relational operators can result in it being pushed down, leading to significant performance benefits. Some database engines have been implemented showing significant benefits by including the skyline or winnow operators [6,8,31,32,38].

SPARQL has been extended with qualitative preferences [41,44,47] based on the winnow operator [11]. Siberski and colleagues [44] propose to add qualitative preference capabilities to SPARQL, focusing on the feasibility of the process. They incorporate the PREFERRING clause into the SPARQL syntax where preferences are separated by the AND construct. In addition, the CASCADE keyword allows for prioritizing a preference criterion over another one. They extended the ARQ query engine [2] with BNL as a proof of concept. However, they do not deal with query processing/optimization issues. SPREFQL [47] is another extension of SPARQL for qualitative preferences. Their work comes nearer to Chomicki's framework [11] than Siberski et al.'s framework [44] because it allows the expression of extrinsic preferences whose formulas may refer both to built-in predicates on the basis of tuples and to other constructors such as database relations. Unlike Siberski et al.'s work [44], they support conditional preferences (if-then-else). At the implementation level, they presented a query rewriting technique that maps from a SPREFQL query to an equivalent SPARQL query by means of the NOT EXISTS operator. They experimentally study the performance of Nested-Loop (NL) algorithm, Block-Nested Loop (BNL), and a query rewriting technique to evaluate preference queries. BNL iteratively compares each instance with non-dominated instances in a window, and the instance is returned only if it is not dominated by any other while NL is a naive algorithm that compares each input instance against all input instances and whose computational complexity is quadratic. Unfortunately, their solution based on query rewriting does not work correctly due to the fact that it is based on SPARQL NOT EXISTS, which has many known problems [40]. Thus, Patel-Schneider and colleagues [41] identified and fixed the problem in the previous proposals for acyclic and transitive preference relations. Moreover, they proposed an efficient implementation for the preference query evaluation using the union-find algorithm [46]. Datalog +/− was extended by Lukasiewicz and colleagues [36] to include preferences and the authors developed algorithms to answer preference queries over Datalog +/− ontologies. In contrast to our work, all these strategies evaluate the preference operator on top of the engine.

OBDA engines are data integration systems usually focused on the transformation of the original sources into a global schema (ontology) [42]. The process can be performed by its corresponding materialization [18] (i.e., transforming input sources to RDF knowledge graphs) but also on query translation techniques for highly dynamic data sources [7,43], where mapping rules are used to translate the input SPARQL query to an equivalent one, usually in SQL [10]. To cover the features of different data formats, many different types of OBDA mapping languages have also been proposed in the last few decades, with a wide variety of syntax and formats. Since its W3C recommendation in 2012, R2RML [17] has become the standard mapping specification for accessing relational databases. Many tools support these rules; some of them materialize the data into a knowledge graph (e.g. DB2Triples[1] and R2RML-Parser[2]) and others provide virtual RDF views, focusing on formalizing the translation of SPARQL into SQL and optimizing the resulting SQL query (Morph-RDB [43] and Ontop [7]). Despite the efforts in the development of these engines, to the best of our knowledge, current existing OBDA engines do not support SPARQL qualitative

[1] https://github.com/antidot/db2triples
[2] https://github.com/nkons/r2rml-parser

queries. The most related work is the one presented by Straccia [45] which considers the problem of evaluating top-k queries in the context of OBDA over relational databases.

## 4. OBDA-based qualitative preference queries

In this section, we describe Morph-Skyline++, an OBDA framework for translating and executing qualitative preference queries from SPARQL-to-SQL. First, we formally define the problem of translating qualitative preference queries from SPARQL-to-SQL. Second, we probe the correctness of translating qualitative preferences from SPARQL-to-SQL. Third, we propose a solution based on the Chebotko and colleagues' translation approach [10] which is implemented in Morph-Skyline++.

### *4.1. Problem definition*

Our formalization is based on OBDA [42], which relies on conceptually representing a domain of interest over data stored in an underlying database system.

**Definition 1** (OBDA Specification [42])**.** OBDA is defined as a triple $\tau = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ where $\mathcal{O}$ is an ontology that describes the domain, $\mathcal{S}$ is a schema, and $\mathcal{M}$ is a mapping between $\mathcal{O}$ and $\mathcal{S}$. In addition, an OBDA instance is defined as a tuple $\mathcal{PI} = \langle \tau, \mathcal{D} \rangle$ where $\tau$ is an OBDA specification and $\mathcal{D}$ is a data instance conforming to $\mathcal{S}$.

A qualitative preference query can be specified on an ontology $\mathcal{O}$ or on a relational schema $\mathcal{S}$. A qualitative preference query returns a set of non-dominated solutions to a SPARQL pattern (resp. a set of non-dominated tuples to a SQL query on a relational schema) where a solution (resp. a tuple) is non-dominated if there is no other solution (resp. other tuple) that is better to it according to a preference relation $\prec$.

**Definition 2** (Dominance)**.** Let $\Sigma$ be a set of solutions to a SPARQL graph pattern $gp$ (resp. a set of tuples to a SQL query $s$ over $\mathcal{S}$) and let $\prec$ be a preference relation over $\Sigma \times \Sigma$. A solution (resp. a tuple) $\delta_1 \in \Sigma$ is dominated by a solution (resp. a tuple) $\delta_2 \in \Sigma$ if $\delta_2 \prec \delta_1$.

A preference relation $\prec$ can be atomic or combined between them as shown in Definition 3. It is important to highlight that a preference relation $\prec$ is a partial order.

**Definition 3** (Atomic and Combined Preference Relations)**.** A preference formula $U(\delta_1, \delta_2)$ is a first-order formula defining a preference relation $U$ in the standard sense, namely, $\delta_1 \prec_U \delta_2$ if $U(\delta_1, \delta_2)$ holds. Let $\prec_U$, $\prec_V$ be two preference relations over $\Sigma \times \Sigma$; we define atomic and combined preference relations as follows:

- Boolean: $\delta_1 \prec_{BE} \delta_2 \equiv BE(\delta_1) \wedge \neg BE(\delta_2)$ where $BE$ is a Boolean expression
- Scoring: $\delta_1 \prec_{\text{LOWEST},<} \delta_2 \equiv \delta_1 < \delta_2$ and $\delta_1 \prec_{\text{HIGHEST},<} \delta_2 \equiv \delta_2 < \delta_1$
- Multidimensional (Skyline): $\delta_1 \prec_{U \otimes V} \delta_2 \equiv \delta_1 \preceq_U \delta_2 \wedge \delta_1 \preceq_V \delta_2 \wedge (\delta_1 \prec_U \delta_2 \vee \delta_1 \prec_V \delta_2)$
- Prioritized: $\delta_1 \prec_{U \rhd V} \delta_2 \equiv (\delta_1 \prec_U \delta_2) \vee (\neg(\delta_1 \prec_U \delta_2) \wedge \neg(\delta_2 \prec_U \delta_1) \wedge (\delta_1 \prec_V \delta_2))$.
- Conditional: $\delta_1 \prec_{\Longrightarrow_{\kappa,\phi,\gamma}} \delta_2 \equiv (\kappa(\delta_1) \wedge \kappa(\delta_2) \wedge \delta_1 \prec_\phi \delta_2) \vee (\neg\kappa(\delta_1) \wedge \neg\kappa(\delta_2) \wedge \delta_1 \prec_\gamma \delta_2)$ where $\kappa$ is a Boolean expression, and $\phi$ and $\gamma$ are preferences.

For our motivating example, preferences for the SPARQL query $Q_1$ can be expressed as the preference formula $C$ in Fig. 3 where preferences are ordered ascending; i.e., that a lower value indicates a higher preference.

**Definition 4** (Qualitative Preference Queries)**.** A qualitative preference query $Q$ over an ontology $\mathcal{O}$ (resp. a schema $\mathcal{S}$) is defined as a pair $Q = (gp, \prec)$ (resp. $Q' = (s, \prec)$) which produces a subset of $\Sigma$ such that: $\{p_1 \in \Sigma \mid \neg\exists p_2 \in \Sigma : p_2 \prec p_1\}$.

To compute the desired solutions (resp. tuples) for all preference relations that are acyclic or transitive, a qualitative preference query was extended by Patel-Schneider and colleagues [41].

**Definition 5** (Extended Qualitative Preference Queries)**.** If $\Sigma$ is a finite set of solutions (resp. tuples) and $\prec$ is a preference relation over $\Sigma \times \Sigma$ then qualitative preference query $Q^t$ returns: $\{p_1 \in \Sigma \mid \neg\exists p_2 \in \Sigma : p_2 \prec^* p_1 \wedge \neg(p_1 \prec^* p_2)\}$ where $\prec^*$ is the transitive closure of $\prec$ over $\Sigma$.

$$
\begin{aligned}
&(?price, ?distance, ?s, ?day) \prec_C \\
&(?price', ?distance', ?s', ?day?`) \equiv \\
&(?price, ?distance, ?s, ?day) \prec_{C_1 \otimes C_2 \otimes C_4} (*) \\
&(?price', ?distance', ?s', ?day?`)
\end{aligned}
$$

$$
\begin{aligned}
&(?price, ?distance) \prec_{C_1} (?price', ?distance') \equiv \\
&?price \leqslant ?price' \wedge ?distance \leqslant distance' \wedge \\
&(?price < ?price' \vee ?distance < distance')
\end{aligned}
$$

$$
\begin{aligned}
&(?s) \prec_{C_2} (?s') \equiv (?s \leqslant 1000) > (?s' \leqslant 1000) \\
&\vee(\neg((?s \leqslant 1000) > (?s' \leqslant 1000)) \wedge \\
&\neg((?s' \leqslant 1000) > (?s \leqslant 1000)) \wedge \\
&(?s \geqslant 1700) > (?s' \geqslant 1700))
\end{aligned}
$$

$$
\begin{aligned}
&(?s, ?day) \prec_{C_3} (?s', ?day') \equiv \\
&(?day =' Saturday' \wedge (?s \leqslant 1000) \wedge \\
&?day' =' Saturday' \wedge \neg(?s' \leqslant 1000)) \vee \\
&(?day =' Friday' \wedge ((?s \leqslant 1000) \vee (?s \geqslant 1700)) \wedge \\
&?day' =' Friday' \wedge \neg((?s' \leqslant 1000) \vee (?s' \geqslant 1700)))
\end{aligned}
$$

*(*)where C4 is LOWEST on solutions after applying* $\prec_{C_3}$

Fig. 3. Preference formulas for our motivating query.

*Problem definition* Given an OBDA Specification $\sigma = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ and a SPARQL qualitative preference query $Q = (gp, \prec)$ with an equivalent relational algebra expression $SA$ over an ontology $\mathcal{O}$, the problem of qualitative preference query translation in an OBDA setting is defined as finding an equivalent qualitative preference query $Q' = (s, \prec)$ over $\mathcal{S}$ where $s$ is a SQL query with an equivalent relational algebra expression $RA$ over a relational schema $\mathcal{S}$.

### 4.2. Qualitative preference query translation

A query can be seen as a set of operators and operands, and two queries are equivalent if they are composed of semantically equivalent operators applied on equivalent data in the same sequence. In this section, we define Semantic Algebras and their corresponding evaluation functions for both SPARQL and SQL based on Relational Algebra operators, since Relational Algebra forms the basis for query translation and its close relationship with SQL. We define its semantics by using tuple relational calculus formulas and the work presented by Codd [15] which proved the equivalence between relational algebra and relational calculus.

**Definition 6** (Semantic Algebra for SPARQL)**.** Given:

- RDF terms $T$ that comprise of pairwise disjoint infinite sets of $I$, $B$ and $L$ (IRIs, Blank nodes and Literals, respectively) from $\mathcal{O}$.
- A partial function $\mu : V \rightarrow T$ that assigns RDF terms of an RDF graph to variables of a SPARQL query where V is an infinite set of variables that is pairwise disjoint from the sets $I$, $B$, and $L$.
- Two mappings $\mu_1$, $\mu_2$ are compatible when for all $x \in dom(\mu_1) \cap dom(\mu_2)$, it is the case that $\mu_1(x) = \mu_2(x)$ where $dom(\mu)$ is the subset of $V$ where $\mu$ is defined.
- A set of mappings $\mu$ or a domain mapping set $\Omega$ where $\Omega \equiv [\![tp]\!]$, and $[\![tp]\!]$ is an evaluation function that maps an element of $gp$ according to Definition 7.
- A SPARQL graph pattern $gp$ defined by the following grammar: $gp ::= tp | gp$ FILTER expr $|$ $gp$ UNION $gp|gp$ AND $gp|gp$ OPT $gp|gp$ PREFERRING crit.

The following operators are defined:

1. $\sigma_{expr}\Omega = \{\mu | \mu \in \Omega \wedge \mu \models expr\}$ where $\mu \models expr$ is evaluated according to Chebotko and colleagues's work [10].

2. $\Pi_{v_1,...,v_n}\Omega = \{\mu_{|v_1,...,v_n} | \mu \in \Omega\}$ where $\mu_{|v_1,...,v_n}$ is a mapping such that $dom(\mu_{|v_1,...,v_n}) = dom(\mu) \cap \{v_1, \ldots, v_n\}$ and $\mu$ and $\mu_{|v1,...,vn}$ are compatible
3. $\Omega_1 \cup \Omega_2 = \{\mu | \mu \in \Omega_1 \vee \mu \in \Omega_2\}$
4. $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 | \mu_1 \in \Omega_1, \mu_2 \in \Omega_2$ are compatible mappings$\}$
5. $\Omega_1 \setminus \Omega_2 = \{\mu \in \Omega_1 |$ for all $\mu' \in \Omega_2$, $\mu$ and $\mu'$ are not compatible$\}$
6. $\Omega_1 \rtimes \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$
7. For the preference operator $\varrho$, $\varrho_{crit}\Omega = \{\mu | \mu \in \Omega \wedge \neg\exists \mu_1 \in \Omega : \mu_1 \prec \mu\}$ where $crit$ is a set of criteria that represents the preference relation $\prec$.

**Definition 7** (Evaluation functions over SPARQL queries)**.** $[\![]\!] : gp \rightarrow \Omega$ denotes an evaluation function that maps an element of $gp$ and an ontology $\mathcal{O}$ to $\Omega$, and given a triple pattern $tp$, $\mu(tp)$ denotes the triple obtained by replacing the variables in $tp$ according to $\mu$.

1. $[\![tp]\!] = \{\mu | dom(\mu) = \text{var}(tp) \wedge \mu(tp) \in \mathcal{O}\}$
2. $[\![gp \text{ FILTER expr }]\!] = \sigma_{expr}[\![gp]\!]$
3. $[\![gp_1 \text{ UNION } gp_2]\!] = [\![gp_1]\!] \cup [\![gp_2]\!]$
4. $[\![gp_1 \text{ AND } gp_2]\!] = [\![gp_1]\!] \bowtie [\![gp_2]\!]$
5. $[\![gp_1 \text{ OPT } gp_2]\!] = [\![gp_1]\!] \rtimes [\![gp_2]\!]$
6. $[\![gp \text{ PREFERRING crit }]\!] = \varrho_{crit}[\![gp]\!]$

**Definition 8** (Semantic Algebra for Relational Algebra)**.** Given:

- An expression $E$ in relational algebra defined by the following grammar: $E ::= \sigma_{Cond}E | \Pi_{A_1,...,A_n}E | E_1 \cup E_2 | E_1 \bowtie_{Cond} E_2 | E_1 \times E_2 | E_1 - E_2 | E_1 \cap E_2 | PREFERRING_{crit}$
- A domain relation, $R = \{t | \xi(t) \equiv \xi(R)\}$ with all tuples such that $t$ and $R$ are defined over the same schema in $\mathcal{S}$ where $\xi(R)$ denotes the schema of a relation $R$.

The following operators are defined:

- $\sigma_{Cond}R = \{t | R(t) \wedge Cond(t)\}$; where $Cond(t)$ is a Boolean expression in SQL such that $Cond$ is satisfied by $t$.
- Let $\{A_1, \ldots, A_n\}$ be a subset of attributes of $R$. $\Pi_{A_1,...,A_n}R = \{t.A_1, \ldots, t.A_n | R(t)\}$
- $R_1 \cup R_2 = \{t | R_1(t) \vee R_2(t) \wedge (\xi(R_1) \equiv \xi(R_2))\}$
- Let $A = \{A_1, \ldots, A_m\}$ and $B = \{B_1, \ldots, B_k\}$ be the sets of attributes of $R_1$ and $R_2$. $R_1 \times R_2 = \{t | \forall t_1, \forall t_2 (R_1(t_1) \wedge R_2(t_2) \rightarrow t.A_1 = t_1.A_1 \wedge \cdots \wedge t.A_m = t_1.A_m \wedge t.B_1 = t_2.B_1 \wedge \cdots \wedge t.B_k = t_2.B_k)\}$
- $R_1 \bowtie_{Cond} R_2 = \sigma_{Cond}(R_1 \times R_2)$
- $R_1 - R_2 = \{t | R_1(t) \wedge (\xi(R_1) \equiv \xi(R_2)) \wedge \forall t_2 (R_2(t_2) \rightarrow t \neq t_2)\}$
- $R_1 \cap R_2 = \{t | R_1(t) \wedge R_2(t) \wedge (\xi(R_1) \equiv \xi(R_2))\}$
- $\varrho_{crit}R = \{t | R(t) \wedge \neg\exists t_1 (R(t_1) \wedge t_1 \prec t)\}$; where $crit$ is a SQL expression that represents the preference relation $\prec$.

**Definition 9** (Evaluation functions over Relational Algebra)**.** $[\![]\!]_r : E \rightarrow R$ denotes an evaluation function that maps an element of $E$ to $R$.

1. $[\![\sigma_{Cond}E]\!]_r = \sigma_{Cond}[\![E]\!]_r$
2. $[\![\Pi_{A_1,...,A_n}E]\!]_r = \Pi_{A_1,...,A_n}[\![E]\!]_r$
3. $[\![E_1 \cup E_2]\!]_r = [\![E_1]\!]_r \cup [\![E_2]\!]_r$
4. $[\![E_1 \bowtie_{Cond} E_2]\!]_r = [\![E_1]\!]_r \bowtie_{Cond} [\![E_2]\!]_r$
5. $[\![E_1 \times E_2]\!]_r = [\![E_1]\!]_r \times [\![E_2]\!]_r$
6. $[\![E_1 - E_2]\!]_r = [\![E_1]\!]_r - [\![E_2]\!]_r$
7. $[\![E_1 \cap E_2]\!]_r = [\![E_1]\!]_r \cap [\![E_2]\!]_r$
8. $[\![\varrho_{crit}E]\!]_r = \varrho_{crit}[\![E]\!]_r$

**Definition 10** (Qualitative Preference Query Translation)**.** Given an OBDA Specification $\sigma = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ and a SPARQL qualitative preference query $Q = (gp, \prec)$ over an RDF graph $G$, the problem of qualitative preference query translation in an OBDA setting extends the standard OBDA setting with preference operators and it is defined as the rewrite of query $Q$ into a qualitative preference query $Q' = (s, \prec)$ over $\mathcal{S}$ that meets the following conditions:

- there is a function $f: gp \rightarrow s$ that maps $gp$ to its corresponding SQL query $s$ over $\mathcal{S}$ according to the mappings $\mathcal{M}$ [10]. In addition, Cyganiak [16] has already been proved that $[\![RA]\!]_r \equiv [\![gp]\!]$ for all the traditional operators in Relational Algebra where $RA$ is the relational algebra expression of SQL.
- there is a mapping $f_i = (term(v), v') \in \mathcal{M}$ such that each preference in $\prec$ of $Q$ matches a preference in $\prec$ of $Q'$, there is no preference in $\prec$ of $Q'$ that does not match a preference in $\prec$ of $Q$, each variable $v$ in the preferences $\prec$ of $Q$ matches a database attribute $v'$ in the preferences $\prec$ of $Q'$ and there is no database attribute in $\prec$ of $Q'$ that does not match a variable in $\prec$ of $Q$ where $term(v)$ is the ontology term corresponding to the variable $v$, e.g., the ontology term associated with the ?price variable is mapped to the database attribute price by using $\mathcal{M}$. Thus, in our motivating example, `LOWEST ?price AND LOWEST ?dist AND (?s <= 1000) PRIOR TO (?s >= 1700) AND LOWEST IF (?day = "Saturday") THEN (?s<=1000) ELSE IF (?day = "Friday") THEN (?s >= 1700 || ?s<=1000) ELSE (false)` is translated to `LOWEST price AND LOWEST distance AND (starts<=1000) PRIOR TO (starts>=1700) AND LOWEST IF (day = "Saturday") THEN (starts <= 1000) ELSE IF (day= "Friday") THEN (starts >= 1700 || starts <= 1000) ELSE (false).`
  For the following two relational algebra expressions of SPARQL and SQL, $SA$ and $RA$, respectively:

$$SA: \varrho_{crit}\Omega = \{\mu | \mu \in \Omega \wedge \neg\exists \mu_1 \in \Omega : \mu_1 \prec \mu\}$$

$$RA: \varrho_{crit}R = \left\{t | R(t) \wedge \neg\exists t_1 \left(R(t_1) \wedge t_1 \prec t\right)\right\}$$

  If $\Omega \equiv [\![tp]\!]$, $crit \equiv \prec$, and $R \in \mathcal{S}$ such that $tp = ?s\ R\ ?o$ then $SA \equiv RA$.
- There is a method to translate $Q$ to $Q'$ where the results of $Q'$ are equal to $Q_{sql}$ being $Q_{sql}$ a perfect SQL query skyline (gold standard), i.e., over any relational database, a SQL query written manually by an expert and the equivalent one in SPARQL (that then, is translated with our proposal) have the same results.

*Proposed solution* We propose Morph-Skyline++ as an OBDA qualitative preference query translation engine for relational databases. Morph-Skyline++ is based on the Chebotko et al.'s translation approach [10] which formalizes query translation from SPARQL into SQL exploiting the use of a set of mapping rules and their corresponding functions. More specifically, Morph-Skyline++ relies on the formal definition of mappings and functions with R2RML provided by Priyatna and colleagues [43] and it exploits the preference operators in databases during query processing. When a SPARQL qualitative preference query is received by Morph-Skyline++, our solution, QualQT uses R2RML mappings to translate the query into SQL. Then, the translated query is executed on a qualitative-preference-enabled query engine and the results are returned.

QualQT is a technique based on a physical preference operator. By using QualQT, a SPARQL qualitative preference query $Q = (gp, \prec)$ is translated into a SQL qualitative preference query $Q' = (s, \prec)$. First, $gp$ is translated to $s$ by means of the function *translate* defined as *trans* by Priyatna and colleagues [43]. Then, each variable $v \in \prec$ from $Q$ is matched to $v' \in \prec$ from $Q'$ according to the mapping $\mathcal{M}$. Thus, the translated qualitative preference query will be executed within the underlying database system. The SQL qualitative preference query in Fig. 1(b) is the result of translating $gp$ to $s$, and each variable in $\prec$.

Algorithm 1 sketches the algorithm QualQT implemented by Morph-Skyline++ to process qualitative preference queries on virtual knowledge graphs. QualQT receives a set of mappings $\mathcal{M}$ and a SPARQL qualitative preference query $Q$. It firstly separates the preferring clause from the query $Q$ and gets a non-qualitative-preference subquery (lines 1-2). In line 3, QualQT translates the non-qualitative-preference query $gp$ into SQL by means of the function *translate* defined as *trans* by Priyatna and colleagues [43]. Line 4 builds a hash structure from the graph patterns belonging to $gp$; this hash structure contains the attribute that maps each variable. Note that *mapping_variables* returns the column/constant that corresponds to each variable in a SPARQL query [43]. The function *mapping_variables* is a function whose input is a graph pattern $gp$, and a set of all possible R2RML mappings $\mathcal{M}$, and its output is a set of relational attributes. Subsequently, for each variable in the preferring clause, QualQT iteratively obtains its corresponding attribute by searching for it in the structure hash and creates the corresponding expression with the mapped attribute (lines 5–7). Each variable in the preferring clause of Fig. 1(a), `LOWEST ?price AND LOWEST ?dist AND (?s<=1000) PRIOR TO (?s >= 1700)`

**Algorithm 1** Qualitative preference query translation, QualQT: $\mathcal{M}$ – mappings, $Q$ – qualitative preference query

1: pref ← preferring clause from $Q$
2: $gp \leftarrow Q \setminus$ pref
3: query ← translate(gp, $\mathcal{M}$)
4: $hash \leftarrow mapping\_variables(gp, \mathcal{M})$
5: **for** each variable $variable_i \in$ pref **do**
6: $\quad attribute_i \leftarrow$ get($hash$, $variable_i$)
7: $\quad$ replace $variable_i$ in pref with $attribute_i$
8: **return** (query, pref)

```
<SolutionModifier> ::= [ <GroupClause> ] [ <HavingClause> ] [ <PreferClause> ]
                       [ <OrderClause> ] [ <LimitOOffsetClauses> ]
<PreferClause> ::= 'PREFERRING' <Preference>
<Preference> ::= <PrioritizedPref> \{'AND' <PrioritizedPref>\}
<PrioritizedPref> ::= <ConditionalOrAtomicPref> \{'PRIOR TO' <ConditionalOrAtomicPref>\}
<ConditionalOrAtomicPref> ::= <ConditionalPref> | <AtomicPref>
<ConditionalPref> ::= 'IF' <Expression> 'THEN' <ConditionalOrAtomicPref> 'ELSE'
                      <ConditionalOrAtomicPref>
<AtomicPref> ::= <Expression> | <HighestPref> | <LowestPref> | <DiffPref>
<HighestPref> ::= 'HIGHEST' <Expression>
<LowestPref> ::= 'LOWEST' <Expression>
<DiffPref> ::= 'DIFF' <Expression>
```

Grammar 1. Grammar for qualitative preferences in SPARQL

`AND LOWEST IF (?day="Saturday") THEN (?s<=1000) ELSE IF (?day = "Friday") THEN (?s >= 1700 || ?s <= 1000) ELSE (false)` is replaced by its corresponding attribute to `LOWEST price AND LOWEST distance AND (starts <= 1000) PRIOR TO (starts >= 1700) AND LOWEST IF (day="Saturday") THEN (starts <= 1000) ELSE IF (day = "Friday") THEN (starts >= 1700 || starts <= 1000) ELSE (false)`. Finally, QualQT returns a pair with the translated query and preferring clause.

## 5. Query language syntax

In this section, we summarize the grammar supported by the parser of Morph-Skyline++. We rely on the PrefSPARQL grammar [25] to support qualitative preferences in SPARQL queries. The grammar basis is SPARQL 1.1, but the definition of <SolutionModifier> changes. Qualitative preferences are specified as a new solution modifier in the EBNF grammar shown in Grammar 1. This grammar includes rules for skyline, prioritized and conditional preferences. The keyword 'AND' in <Preference> allows for separating independent dimensions of skyline queries, and the rules <HighestPref>, <LowestPref> and <DiffPref> correspond to maximizing, minimizing, and grouping criteria in skyline queries. Unlike previous works [25,41,44,47], we have included the DIFF directive from Börzsönyi and colleagues's work [6]. Also, Prioritized and Conditional (IF-THEN-ELSE) preferences are included in the grammar as the rules <PrioritizedPref> and <ConditionalPref>, respectively. In addition, all non-terminals that are not defined in this grammar are defined by the standard SPARQL syntax. Finally, with respect to the extended qualitative preference queries, they can be specified by means of recursive queries in SPARQL. Section 5.2 briefly describe how to express extended qualitative preference queries in SPARQL.

### *5.1. Translating qualitative preference queries into SQL*

In SQL, Preference SQL [38] is a query language to express user wishes. Grammar 2 presents the grammar corresponding to the preferring clause for Preference SQL. The preferring clause must be before the GROUP BY

```
<PreferClause> ::= 'PREFERRING' <Preference> ['PARTITION' 'BY' <Expression>\{','<Expression>\}]
<Preference> ::= <PrioritizedPref> \{'PLUS' <PrioritizedPref>\}
<PrioritizedPref> ::= <ConditionalOrAtomicPref> \{'PRIOR TO' <ConditionalOrAtomicPref>\}
<ConditionalOrAtomicPref> ::= <ConditionalPref> | <AtomicPref>
<ConditionalPref> ::= 'CASE' <Expression> \{'WHEN' <ConditionalOrAtomicPref> 'THEN' <result>\}
                      'ELSE' <ConditionalOrAtomicPref>
<AtomicPref> ::= <Expression> | <HighestPref> | <LowestPref> | <DiffPref>
<HighestPref> ::= 'HIGH' <Expression>
<LowestPref> ::= 'LOW' <Expression>
```

Grammar 2. Grammar for qualitative preferences in PREFERENCE SQL

```
SELECT *
FROM laboratory l, offers o, appointment a
WHERE l.lab_id = o.lab_id AND o.app_id = a.app_id
      AND NOT EXISTS (
            SELECT *
            FROM laboratory l1, offers o1, appointment a1
            WHERE l1.lab_id = o1.lab_id AND o1.app_id = a1.app_id AND
              -- Multidimensional preferences
              o1.price<=o.price AND o1.distance<=o.distance AND (o1.price<o.price OR o1.distance<=o.distance)
              -- Prioritized preferences
              AND ( (o1.starts<=1000) > (o.starts<=1000) OR (NOT ((o1.starts<=1000) > (o.starts<=1000)) AND
                                                             NOT ((o.starts<=1000) > (o1.starts<=1000)) AND
                                                              (o1.starts >= 1700) > (o.starts >= 1700))) AND
              -- conditional preferences
              ((o1.day = "Saturday" AND o.day = "Saturday" AND (o1.starts <= 1000) > (o.starts <= 1000)) OR
               (o1.day = "Friday" AND o.day = "Friday" AND ((o1.starts <= 1000) > (o.starts <= 1000)) OR
               (o1.starts >= 1000) > (o.starts >= 1000)) ))
```

Fig. 4. A sample SQL translated qualitative preference query for booking a PCR appointment. The query identifies cheaper and closer laboratories with earlier appointments before 10:00 or after 17:00 on Friday or Saturday by means of a nested SQL query.

clause and after the WHERE clause. The partition-by clause can be seen as the DIFF directive in a skyline query. In contrast to PrefSPARQL, PLUS, CASE-WHEN-THEN-ELSE, HIGH, and LOW are used instead of AND, IF-THEN-ELSE, HIGHEST, and LOWEST, respectively.

In both SQL and SPARQL, the keywords `AND`, `PRIOR TO`, and `IF-THEN-ELSE` (resp. `CASE-WHEN-THEN-ELSE`) are used for the multidimensional, prioritized, and conditional preference relations, respectively.

Lastly, a qualitative preference query can be translated into a nested SQL query which compares each tuple with each other tuple. `Q PREFERRING` $\prec_{Pref}$ can be expressed in SQL as `Q WHERE NOT EXISTS (Q' trans(Q,Q',`$\prec_{Pref}$`))` where `Q'` is the query `Q'` but with the same tables renamed with aliases, $\prec_{Pref}$ the preference criteria, and `trans` performs pairwise dominance checks for each pair of tuples. The function `trans` is recursively applied as:

- for $\texttt{Pref} = \texttt{Pref}_1$ `AND` $\texttt{Pref}_2$, $\texttt{trans}(Q,Q',\prec_{Pref}) = \texttt{trans}(Q,Q',\prec_{Pref_1 \otimes\ Pref_2})$
- for $\texttt{Pref} = \texttt{Pref}_1$ `PRIOR TO` $\texttt{Pref}_2$, $\texttt{trans}(Q,Q',\prec_{Pref}) = \texttt{trans}(Q,Q',\prec_{Pref_1 \rhd\ Pref_2})$
- for `Pref = IF E THEN` $\texttt{Pref}_1$ `ELSE` $\texttt{Pref}_2$, $\texttt{trans}(Q,Q',\prec_{Pref}) = \texttt{trans}(Q,Q',\prec_{\Longrightarrow E,Pref_1,Pref_2})$

Our motivating query in Fig. 1(b) can be translated into a nested SQL query as depicted in the Fig. 4.

In our experimental study, as a baseline, we compare our work against qualitative preference queries in SQL and their translation into nested SQL queries to evaluate how costly it is to perform a qualitative preference query on virtual knowledge graphs. We have omitted a detailed description of the SPARQL query rewriting technique [47] to translate a qualitative preference query from SPARQL to SPARQL. The authors have already shown that this technique does not scale. However, intuitively, the triple patterns within an OPTIONAL clause retrieve the dominated solutions, and similar to a left outer join, those solutions not bound to the dominated ones are in the answer. Our motivating query in Fig. 1(a) can be translated into a nested SPARQL query as depicted in the Fig. 5. For a more detailed description of this technique, please refer to Troumpoukis and colleagues's work [47].

```
SELECT *
WHERE {
      ?l :price ?price; :distance ?dist .
      ?l :offers ?a; :starts ?s; :ends ?e; :day ?day .
      OPTIONAL {
        ?l2 :price ?price2; :distance ?dist2 .
        ?l2 :offers ?a2; :starts ?s2; :ends ?e2; :day ?day2 .
        FILTER(-- Multidimensional preferences
              ?price2<=?price && ?distance2<=?distance && (?price2<?price || ?distance2<=?distance) &&
              -- Prioritized preferences
              ((?s2<=1000) > (?s<=1000) || (!((?s2<=1000) > (?s<=1000)) && !((?s<=1000) > (?s2<=1000)) &&
                                                                    (?s2 >= 1700) > (?s >= 1700))) &&
              -- conditional preferences
              ((?day2 = "Saturday" && ?day2 = "Saturday" && (?s2 <= 1000) > (?s <= 1000)) ||
               (?day2 = "Friday" && ?day2 = "Friday" && ((?s2 <= 1000) > (?s <= 1000)) ||
               (?s2 >= 1000) > (?s >= 1000)))))}
      FILTER(!BOUND(?s2))}
```

Fig. 5. A sample SPARQL translated qualitative preference query for booking a PCR appointment. The query identifies cheaper and closer laboratories with earlier appointments before 10:00 or after 17:00 on Friday or Saturday by means of a nested SPARQL query.

Lastly, knowing the grammar for preferences in SPARQL and SQL, a qualitative preference query can be translated from SPARQL to SQL.

### 5.2. *Expressing extended qualitative preference queries into SPARQL*

An extended qualitative preference query can be implemented by using property paths in SPARQL. Patel-Schneider and colleagues [41] computed the transitive closure of $\prec$ over the solutions. For this, Algorithm 2 first creates a graph $:SG$ with the solutions to the query without preferences (step 1). Subsequently, it constructs another graph $:DG$ by using the previous graph $:SG$ with the solutions dominated by each solution (step 2). Finally, if a solution $s$ in $:SG$ is not transitively dominated by any solution and $s$ does not transitively dominate any solution, then $s$ is returned (step 3). To compute that a solution is transitively dominated (or transitively dominates), property paths are applied.

**Algorithm 2** Extended qualitative preference query mapping, $Q$ - qualitative preference query. Algorithm taken from Patel–Schneider and colleagues [41]

1: For a Qualitative Preference Query, $Q$, in the form of `SELECT V WHERE { P PREFERRING C}`, generate a graph with a "fresh" name :SG in the current dataset containing nodes representing each solution of the original query by a CONSTRUCT query: `CONSTRUCT { _:b a :solution ; :_1` $V^1$`;...;:` $\_nV^n$ `} WHERE { P }` where $V^i$ is the $i$th variable in $V$.

2: Use a CONSTRUCT query to generate a graph with :dominates edges for solution domination and give it a "fresh" name :DG in the current dataset:

```
CONSTRUCT { ?s1 :dominates ?s2 }
WHERE { ?GRAPH :SG { ?s1 a :solution ; :_1 V1_1;...;:_nV1_n .
                     ?s2 a :solution ; :_1 V2_1;...;:_nV2_n . }
        FILTER ( C ) }
```

3: Return the solutions that are not transitively dominated by a solution that they do not transitively dominate:

```
SELECT V FROM NAMED :SG FROM NAMED :DG
WHERE { GRAPH :SG { ?s a :solution ; :_1 V1;...;:_nVn. }
        GRAPH :DG { OPTIONAL { ?s1 :dominates+ ?s .
           OPTIONAL { ?s :dominates+ ?s1. BIND(1 AS ?dd) }
           FILTER ( !bound(?dd) )
           BIND(1 AS ?d)}
        FILTER ( !bound(?d) ) } }
```

Table 1
A summary of experiments

| Section | Objective | Benchmarks | Hypothesis |
|---|---|---|---|
| 6.1 | A performance comparison between SPARQL preference qualitative queries and their equivalent SQL preference qualitative queries | TPC-H | $H_1$, $H_2$, and $H_3$ |
| 6.2 | A performance comparison between Morph-Skyline++ and state-of-the-art triplestores. One of the tools only supports skyline queries. | TPC-H and Linked-MDB | $H_4$ and $H_5$ |
| 6.3 | A preliminary study of the scalability of extended qualitative preference queries | Gas Stations | $H_6$ |

## 6. Experimental study

We study the efficiency and effectiveness of our two implementations of Morph-Skyline++. First, we describe the hypotheses that we want to validate as well as the datasets and queries, mappings, metrics, and implementation details for our experimental study. A summary of our evaluation setup is presented in Table 1. All the resources used are online.[3]

*Hypotheses* The impact of the preference criteria and the dataset size on the execution time are studied over different datasets. Our study aims at evaluating the following hypotheses to test the correctness of Morph-Skyline++ approach (H1–H4) and its capabilities in comparison with the state of the art:

- $H_1$ Morph-Skyline++ incurs extra overhead when executing a preference query versus running the preference query in SQL over a given relational database engine, EXASol.
- $H_2$ As the number of preference criteria increases, the number of solutions increases, and Morph-Skyline++ increases its execution time.
- $H_3$ As the dataset size increases, the answer size also enlarges, and Morph-Skyline++ increases its execution time.
- $H_4$ QualQT executed by Morph-Skyline++ has better runtime than the preference-based operator evaluated on top of triplestores.
- $H_5$ QualQT executed by Morph-Skyline++ performs better at runtime than the current SPARQL-based approaches on materialized RDF for skyline query evaluation.
- $H_6$ As the dataset size increases, the execution time of computing the winnow operator $\omega^t_{\succ}$ grows exponentially.

*Benchmarks and queries* *i*) **TPC-H**: We use the Transaction Processing Council Ad-hoc/decision support benchmark consisting of 15 queries expanded by PREFERRING clauses [38]. We adapt this benchmark to SPARQL queries. TPC-H datasets are generated in terms of scale factors, where a scale factor of 1 corresponds to 1 GB of data. We have generated TPC-H datasets for scale factors of 1, 5, 10, 20, 30, 40, 50, and 60. A summary of TPC-H datasets and queries is presented in Table 2. The 1 GB dataset was transformed into RDF by means of SDM-RDFizer [27] and Table 2 contains the number of triples and the size of TPC-H transformed into RDF. To test the methods brTPF-MT, and SkyTPF-MT [29], we also transform the RDF data to HDT by means of the HDT library.[4] *ii*) **Linked Movie Database (LinkedMDB)**: We run 7 LinkedMDB queries defined by Troumpoukis and colleagues [47] and a summary of this dataset and queries can be observed in Table 3. This dataset was manually transformed to CSV using SQL queries and then it was loaded into an RDB to be evaluated by Morph-Skyline++. *iii*) **Gas Stations**: Patel-Schneider and colleagues [41] presented a running example for gas stations over 18 instances. We have manually converted this data to CSV and then scaled it with VIG [35]. We have created an equivalent table and we have loaded this CSV file into mysql. We have run VIG with different scales: 5,10, 50, 100, 500, and 1000. A scale value $v$ indicates that the table size increases $v$ times. A summary of the number of triples and size for these datasets can be found in Table 3. This dataset will only be used for the experimental study of the extended qualitative preferences queries.

[3] https://github.com/marleeng/morph-preferences
[4] http://www.rdfhdt.org/

Table 2
TPC-H. Number of rows per table for different scales (left) and a summary of queries (right)

| Scale | Lineitem | Orders | Partsupp | Part | Customer | Supplier |
|---|---|---|---|---|---|---|
| 1 GB | 6M | 1.5M | 0.8M | 0.2M | 0.15M | 0.01M |
| 5 GB | 30M | 7.5M | 4M | 1M | 7.5M | 0.05M |
| 10 GB | 60M | 15M | 8M | 2M | 1.5M | 0.1M |
| 20 GB | 120M | 30M | 16M | 4M | 3M | 0.2M |
| 30 GB | 180M | 45M | 24M | 6M | 4.5M | 0.3M |
| 40 GB | 240M | 60M | 32M | 8M | 6M | 0.4M |
| 50 GB | 300M | 75M | 40M | 10M | 7.5M | 0.5M |
| 60 GB | 360M | 90M | 48M | 12M | 9M | 0.6M |

| Preference | Queries |
|---|---|
| Skyline | Q0, Q2, Q4, Q6, Q9, Q12, Q14 |
| Partitioning (DIFF) | Q10–Q14 |
| Prioritized composition | Q1, Q3, Q5, Q7, Q10, Q11 |
| Conditional preferences | Q8, Q13 |

Table 3
LinkedMDB. Queries summary (left) and summary of RDF datasets (right)

| Preference | Queries |
|---|---|
| Skyline | Q1, Q2 & Q4 |
| Partitioning (DIFF) | Q1 |
| Prioritized composition | Q3 & Q5 |
| Conditional preferences (EXISTS) | Q6 & Q7 |

| Dataset | Triples | Size |
|---|---|---|
| TPC-H | 111M | 20 GB |
| LinkedMDB | 6M | 851 M |
| 1-scaled Gas Stations | 18 | 0.54 KB |
| 5-scaled Gas Stations | 100 | 2.70 KB |
| 10-scaled Gas Stations | 200 | 5.30 KB |
| 50-scaled Gas Stations | 1K | 27 KB |
| 100-scaled Gas Stations | 2K | 55 KB |
| 500-scaled Gas Stations | 10K | 285 KB |
| 1000-scaled Gas Stations | 10M | 332 MB |

*Mappings* *i*) For the TPC-H benchmark, we have created an R2RML mapping document for accessing each relational table with 53 PredicateObjectMaps, 53 Predicates, 53 ObjectsMaps, and 9 JoinConditions. *ii*) For LinkedMDB, an R2RML mapping document was built with 5 TriplesMaps, 10 PredicateObjectMaps, 10 Predicates, 10 ObjectsMaps, and 5 JoinConditions.

*Evaluation metrics* We measure performance as query execution time. It is computed as the elapsed time in seconds between the submission of a query to the engine and the delivery of the answers. Each query was executed 5 times in cold mode and timeouts are set up to 1 hour. The quality of preference-based techniques are also measured in terms of precision, recall and F-measure. Precision measures the percentage of instances that should be in the preferred set computed in terms of the ground truth; recall measures the percentage of instances produced in terms of the ground truth. Ground truth corresponds to the preferred set directly retrieved from the relational database engine and then materialized in RDF using the R2RML mappings and the SDM-RDFizer [27] engine to ensure their correctness. We have included these measures because the state-of-art tools gave different answers for some queries.

*Engines* Morph-Skyline++ is written in Scala and extends ARQ Jena. The relational database engine used is EXASol because it includes a dedicated preferences-based operator implementing the BNL algorithm [38]. We have compared Morph-Skyline++ against SPREFQL [47], developed in Java within the RDF4J framework[5] and querying data loaded into a locally deployed SPARQL endpoint of Virtuoso Community Edition Version 7.1. We have also evaluated the multi-threaded version of the skyTPF and brTPF-based methods (brTPF-MT, and SkyTPF-MT, respectively) [29], a Java servlet implementation that uses RDF-HDT data sources to process skyline queries over Bindings-restricted Triple Pattern Fragments (brTPF). We considered testing our work against NL [47], and TPF-based method [29], but the authors showed that these algorithms have worse performance with respect to BNL,

[5] http://rdf4j.org/

and the skyTPF and brTPF-based methods, respectively. Experiments are executed on a machine with the following characteristics: 2 GHz CPU with 8 cores, 64 GB RAM with Ubuntu 18.04 as its operating system.

### *6.1. Cost of exploiting preferences-based operators over OBDA engines*

Based on the state of the art in skyline queries, which demonstrate that the skyline operator integrated into a database engine is much more efficient than the corresponding operator, but evaluated by translating the skyline query into a SQL query on top of it [6,8,38], we wanted to study how expensive a preferences-based operator is in an OBDA engine with respect to the equivalent SQL queries. Thus, our baseline includes a SQL query with preferring clause and a translated SQL query that uses a nested query with NOT EXISTS [6]. We compare the evaluation of the preferences-based queries when the operator is integrated into the RDBMS or when the operator is not supported by the RDBMS and must be translated into an equivalent SQL query to be executed. In this section, we analyze how significant the performance degradation can be if a preferences-based query is directly executed by an OBDA engine.

Figure 6 reports the average execution time in seconds on a log scale for the 15 TPC-H queries. These queries are expressed in SPARQL so that they can be evaluated by Morph-Skyline++ using QualQT, and their equivalent queries are expressed in SQL with either the preferring clause (SQL-Pref) or NOT EXISTS (SQL) to be directly executed by EXASol. In Fig. 6(a), we can observe that Morph-Skyline++ is slightly worse than the SQL query translated with NOT EXISTS for the first query, Q0, which has only 1 criterion, but as the number of criteria increases (Q1-Q9), the Morph-Skyline++ runtime remains between the preferences-based operator runtime (SQL-Pref) and the evaluation time of the SQL query translated with NOT EXISTS. Morph-Skyline++ is up to two orders of magnitude less than the SQL query translated with NOT EXISTS. These results indicate us that the SPARQL-to-SQL translation and data conversion from RDB to triples does not have a high impact on the performance of Morph-Skyline++ and even Morph-Skyline++ has a better performance than the evaluation of the SQL query translated with NOT EXISTS. For the queries Q10-Q14, Morph-Skyline++ is the worst and even throws timeouts in the last two queries. In particular, these queries are the ones that produce the most results because they identify the preferred solutions per group, i.e., they are grouping queries. To get some idea on the number of results, while queries Q0-Q9 return less than 73 solutions, queries Q10-Q12 return between 10K and 20K solutions approximately, and queries Q13-Q14 return between 58K and 102K solutions approximately. When we analyzed the worst performance of Morph-Skyline++ for these queries, we have observed that the translation time was very small, the SQL query execution time is already included in Fig. 6(a), but the data conversion from the RDB was the process that required significantly more time. The generation of results by Morph-Skyline++ is a costly process since it produces the triples in XML format. For most cases, the Morph-Skyline++ time is of the same order of magnitude as the execution time of SQL-Pref except for cases where the results generated are over 50K solutions. **Therefore, our hypothesis $H_1$ holds: Morph-Skyline++ has additional overhead w.r.t the equivalent preference SQL query execution by an RDB engine**.

Figures 6(b)–6(d) depict the average execution time in seconds on a log scale for the 15 TPC-H queries over dataset sizes from 5 GB to 20 GB. In order of query complexity, the most complex queries are Q13 and Q14 because they are grouping ones and they have four criteria. After those queries, the queries Q7–Q9 are the next most complex. They have four criteria, and the queries Q8 and Q9 are the same as the queries Q13 and Q14 but without grouping. As the dataset size and the query complexity increase, it becomes more difficult to answer the query and timeouts occur. The grouping, number of criteria and conditionals increase the query complexity and as a result, the time for query execution. For dataset sizes greater than 10 GB, timeout also occurs in the queries Q13–Q14 for the SQL query translated with NOT EXISTS and SQL-Pref. Also, Fig. 7 shows the average query time varying the dataset sizes for several configurations of queries. As observed, for the query Q5 characterized by prioritized preferences and skyline, Morph-Skyline++ presents a comparable performance to SQL-Pref, while the SQL query translated with NOT EXISTS significantly worsened due to the increase in the dataset size. For the query Q8 which includes skyline and conditional preferences, Morph-Skyline++ begins to degrade its performance from 10 GB. For the query Q11 with skyline, grouping and prioritized preferences, a timeout occurs from 30 GB for Morph-Skyline++ and the equivalent SQL query evaluations. And the worst case for Morph-Skyline++ and the equivalent SQL query evaluation is the query Q13 which involves skyline, grouping and conditional preferences, and timeouts

Table 4
Precision (P), Recall (R) and F-measure (F). SPREFQL produces incorrect and incomplete answers for 4/15 queries

| Query | Precision | Recall | F-Measure |
|---|---|---|---|
| 7 | 8.451 | 17.143 | 11.321 |
| 9 | 8.276 | 16.438 | 11.009 |
| 10 | 1.250 | 100.000 | 2.469 |
| 14 | 1.704 | 17.518 | 3.105 |

occur from 10 GB. The observed results suggest that Morph-Skyline++ and SQL query evaluations are not able to scale up to complex queries for larger datasets. **Thus, our hypothesis $H_2$ and $H_3$ holds for Morph-Skyline++: its execution time increases with the number of criteria and dataset size because the preferences-based operator of EXASol implements BNL and must perform a greater number of dominance checks to determine whether a solution is preferred or not**.

Figures 6(e)–6(h) show the average execution time in seconds on a log scale for the 15 TPC-H queries over dataset sizes from 30 GB to 60 GB. We can observe that the behavior is similar. For the queries Q0-Q7, the Morph-Skyline++ runtime is between the SQL-Pref runtime and the runtime of the SQL query translated with NOT EXISTS but timeout occurs when the query complexity increases. For dataset sizes greater than 30 GB, timeout also occurs for SQL-Pref and the SQL query translated with NOT EXISTS, and the queries Q10-Q14.

### *6.2. Morph-Skyline++ vs native SPARQL*

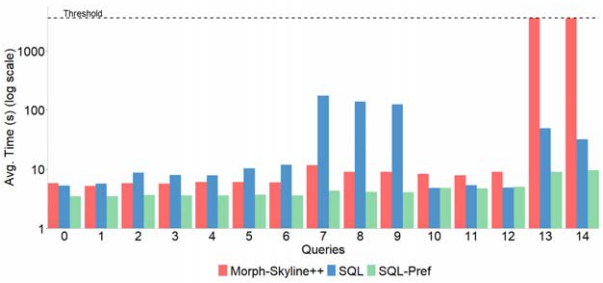
In this section, we compare Morph-Skyline++ against SPREFQL, brTPF-MT, and skyTPF-MT to analyze their performance with respect to the number of criteria for 1 GB TPC-H and LinkedMDB. With this comparison, we want to demonstrate the importance of having specific physical operators integrated into the database engine for preferences-based queries, and that the evaluation of them on top of the database engine are not a good solution neither to native knowledge graphs (RDF) nor virtual knowledge graphs. Within the state-of-the-art tools that support preferences-based SPARQL queries, the preferences-based operator is evaluated on top of triplestores [47] or standard query interfaces such as TPF (Triple Pattern Fragments) [29]. It is noteworthy that although we wanted to compare Morph-Skyline++ with the SPARQL preference query rewriting technique, Morph-Skyline++ was not able to run the SPARQL translated queries on 1 GB TPC-H where the biggest TPC-H table has 6M tuples because Exasol was running out of disk space. Already in our previous work [24], our experimental study showed that from 1M tuples, the translation technique does not scale. The biggest TPCH table for 1 GB has 6M tuples.

Figure 8(a) reports the average execution time in seconds for the 15 TPC-H queries on Morph-Skyline++, BNL of SPREFQL, and the rewriting technique on SPREFQL (RW-S). The first observed result is that RW-S timed out for most queries. RW-S is costly because of expensive not bound, optional and filter clauses. RW-S is not a good option for such a volume of data. The second observed result is that Morph-Skyline++ was the best for the first 13 queries. For the queries Q13 and Q14, timeout occurs because of the execution time of the result generation. It is important to emphasize that SPREFQL was unable to run the queries Q5, Q8, Q11, and Q13. Thus, we have double checked these queries by running them without the preference clause directly in Virtuoso and the result was the successful execution of them; therefore, we think it is because all these queries include functions in their preference clause, in particular, the ABS function, and SPREFQL is unable to parse and process them adequately.

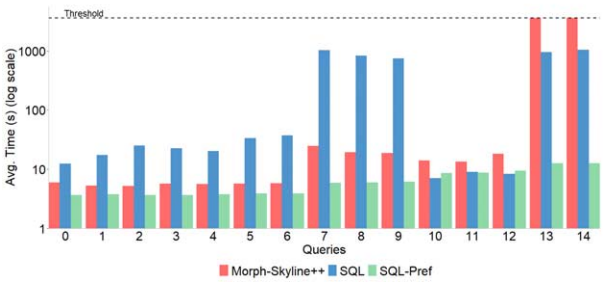
Additionally, we reported on the SPREFQL quality for the 1 GB TPC-H dataset. BNL produces incomplete and incorrect responses for some queries. Table 4 reports precision, recall, and F-measure for SPREFQL for 4 queries. For the remaining queries, precision, recall and F-measure of SPREFQL is 100%. We can observe that precision, recall and F-measure are very low when the number of criteria is high (= 4) or the query is a grouping one.

To compare Morph-Skyline++ with brTPF-MT and skyTPF-MT, Fig. 8(b) shows the average execution times in seconds on a log scale for the skyline queries in TPC-H: Q0, Q2, Q4, Q6, Q8, and Q9. Although skyTPF-MT is one of the best methods presented by Keles and Hose [29], we observed that timeout occurred in 5 queries. This timeout occurs due to the fact that skyTPF-MT continues to work despite a Null Pointer Exception .[6] skyTPF-MT only

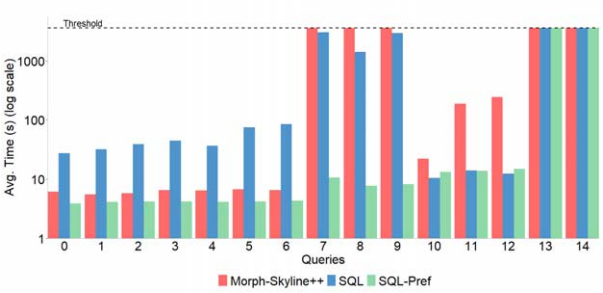
[6]This error has already been reported to the authors.



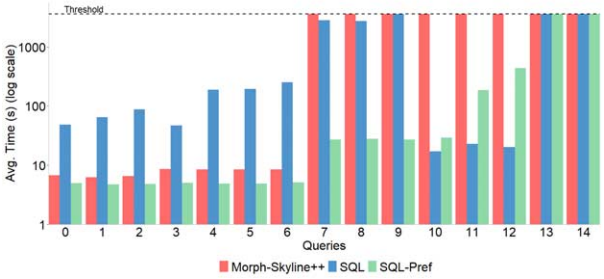

(a) Performance of Morph-Skyline++ for an 1GB dataset



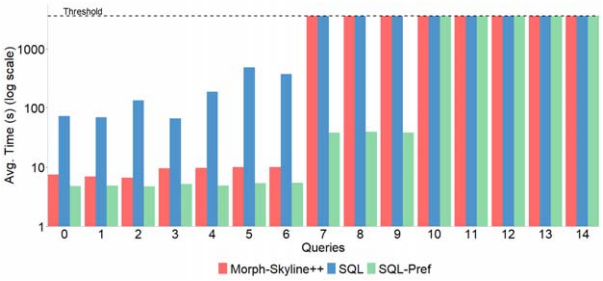

(b) Performance of Morph-Skyline++ for a 5GB dataset



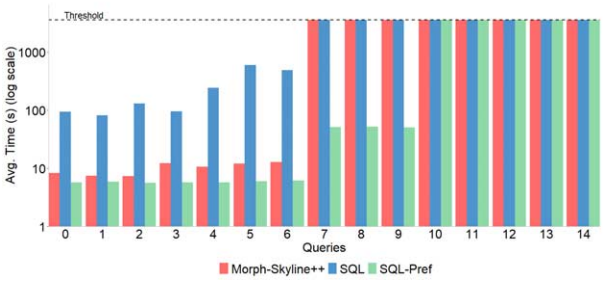

(c) Performance of Morph-Skyline++ for a 10GB dataset



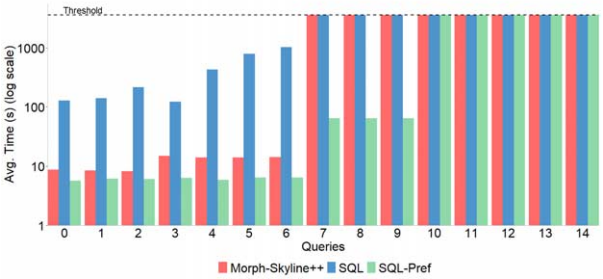

(d) Performance of Morph-Skyline++ for a 20GB dataset



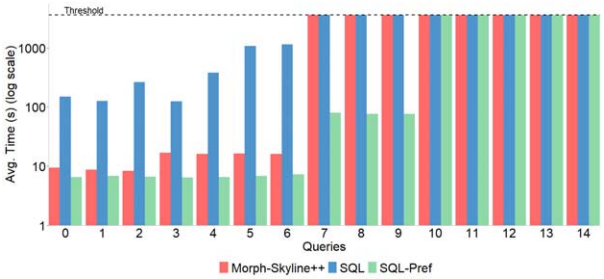

(e) Performance of Morph-Skyline++ for a 30GB dataset




(f) Performance of Morph-Skyline++ for a 40GB dataset

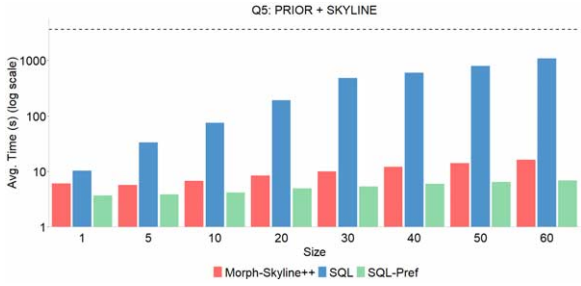



(g) Performance of Morph-Skyline++ for a 50GB dataset

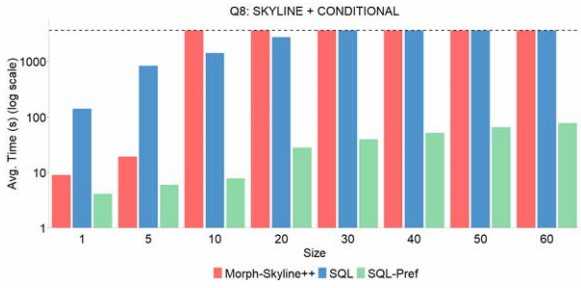



(h) Performance of Morph-Skyline++ for a 60GB dataset

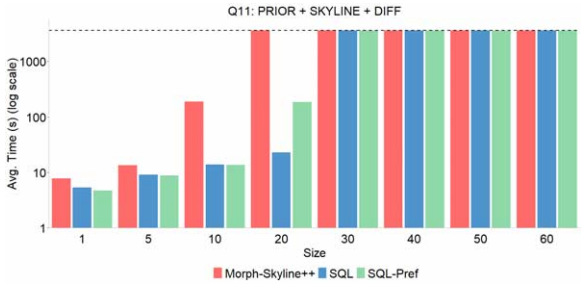
Fig. 6. Performance of Morph-Skyline++ for 1-60GB datasets. We compare QualQT implemented by Morph-Skyline++ against execution of skyline SQL queries and translated skyline SQL queries directly in the database engine for TPC-H in different database sizes.



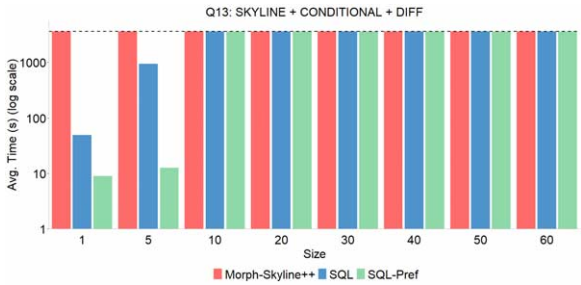

(a) Performance of Morph-Skyline++ for the query Q5




(b) Performance of Morph-Skyline++ for the query Q8



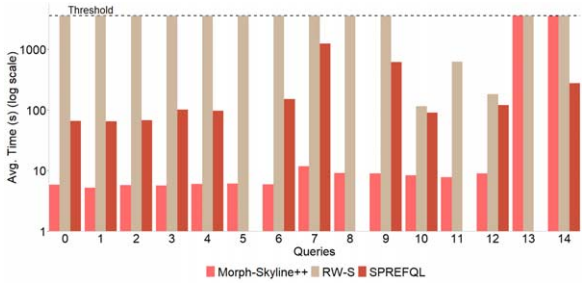

(c) Performance of Morph-Skyline++ for the query Q11



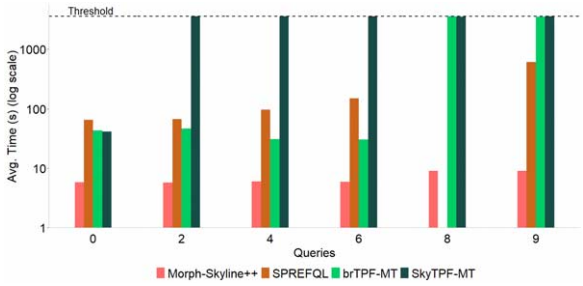

(d) Performance of Morph-Skyline++ for the query Q13

Fig. 7. Performance of Morph-Skyline++. We compare QualQT implemented by Morph-Skyline++ against execution of skyline SQL queries and translated skyline SQL queries directly in the database engine for different queries.

successfully ran for the first query. And, once more, Morph-Skyline++ is the clear winner. **Thus, our hypothesis $H_4$ and $H_5$ hold: the physical operator of Morph-Skyline++ executed within a relational database engine has better performance in view of the fact that the engine optimizes the query.** The execution time for Morph-Skyline++ is at least one order of magnitude less than RDF tools. Given the time required by the RDF tools, it




(a) Performance of Morph-Skyline++ and SPREFQL




(b) Performance of Morph-Skyline++, SPREFQL and skyTPF for skyline queries

Fig. 8. Performance of Morph-Skyline++. We compare QualQT implemented by Morph-Skyline++ against SPREFQL, and TPF, brTPF and skyTPF-methods for the 1 GB TPC-H dataset.

Table 5
Precision (P), Recall (R) and F-measure (F). brTPF and skyTPF-methods produce incorrect and incomplete answers for 4/15queries

| Query | Precision | Recall | F-Measure |
|---|---|---|---|
| 0 | 100.00 | 16.66 | 28.57 |
| 2 | 100.00 | 100.00 | 100.00 |
| 4 | 0.00 | 0.00 | 0.00 |
| 6 | 48.15 | 39.39 | 43.33 |

would be an added value to incorporate the preferences-based operator within the RDF engine.

In addition, we reported on the quality of the methods presented by Keles and Hose [29] for the 1 GB TPC-H dataset. These methods produce incomplete and incorrect responses. Table 5 reports precision, recall and F-measure for brTPF-MT and skyTPF-MT for 4 queries. For the remaining queries, precision, recall and F-measure are not reported because of timeout. The low recall of the queries Q0 and Q6 is due to the fact that these methods produce different answers and a lower number of instances. With these results, we presume that the methods have problems with duplicated data in TPC-H.

Finally, we also compare Morph-Skyline++ and SPREFQL on LinkedMDB. In this case, we can run the rewriting technique for Morph-Skyline++ (RW-M) because the data is smaller. Figure 9 reports the average execution time in seconds for the 7 LinkedMDB queries on Morph-Skyline++ and SPREFQL. It should be highlighted that we do not report the times of the methods TPF-MT, brTPF-MT, and skyTPF-MT because they did not produce results. For this benchmark, the BNL algorithm of SPREFQL has the best performance, although Morph-Skyline++ has the same order of magnitude as SPREFQL. The number of results returned by queries is very low and evidently, SPREFQL efficiently processes the data through access mechanisms of Virtuoso. Additionally, the rewriting technique has the worst performance w.r.t BNL and Morph-Skyline++. However, the rewriting technique of SPREFQL (RW-S) becomes better than Morph-Skyline++ for the queries Q2, Q3, Q6, and Q7. The queries Q2, Q3, and Q6 are queries with instantiations, i.e., the basic graph patterns use constants while the queries Q6 and Q7 only have

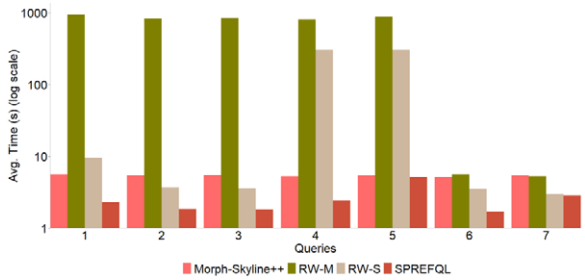



Fig. 9. Performance of Morph-Skyline++ and SPREFQL. We compare Morph-Skyline++, SPREFQL and the rewriting technique on Morph-Skyline++ and SPREFQL for the 7 LinkedMDB queries.

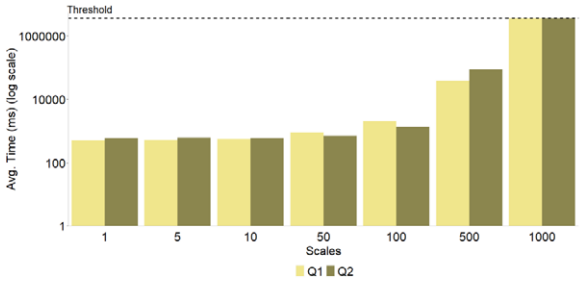



Fig. 10. Performance of $\omega^t_{\succ}$. We report the performance of $\omega^t_{\succ}$ by using recursive queries directly in virtuoso on dataset sizes from 1 to 1000.

a simple criterion with the EXISTS condition. Endris and colleagues [21] empirically show that RDF engines may be better than RDB ones when the query has instantiations. With reference to the rewriting technique on Morph-Skyline++ (RW-M), we have analyzed the execution plans of this kind of query on EXASol. The main complex part of the query plan is a left outer join between the input table $R$ and the result produced by a theta self-join of the input table $R : R \times (R \times R)$. The engine, after performing the theta self-join operator, builds an on-the-fly index for the left outer join to optimize the execution. This index can be large, e.g., the index comprises approximately 200 million entries for 100K tuples and a 4-dimensional query. Finally, EXASol executes a left outer join using this index to discard the dominated ones, i.e., all tuples from $R$ minus matched tuples from the theta self-join (dominated ones); the resulting tuples are those ones that have NULL in the case of no matching join predicate. As a consequence, RW-M performance is the worst. In addition, we can observe in Fig. 9 that for the last two queries, times of the rewriting techniques, RW-M and RW-S, are slightly worse than BNL and Morph-Skyline++, respectively because the rewriting is simpler and does not require additional self-joins to express the equivalent query, i.e., these queries only check that the criterion is true by means of an exists predicate.

### 6.3. Recursion

In this section, we present a preliminary experimental study on the winnow operator $\omega^t_{\succ}$. This winnow operator is evaluated by computing the transitive closure of a preference relation over candidate solutions. We have evaluated the method that computes $\omega^t_{\succ}$ by means of recursive queries. Since recursive queries are not supported by Morph-RDB, we have evaluated the equivalent preferences-based SPARQL query directly in Virtuoso. Figure 10 shows how the query performance is affected by scaling the dataset size from 1 to 1000. We can observe that the execution time grows exponentially as the dataset size increases. The query runtime is clearly affected by the dataset size.

### 6.4. Discussion

In this section, we have considered the performance issues of implementing a preference-based operator as a physical operator in an OBDA engine. Although the preference-based operator can be expressed in SPARQL [47],

a rewriting of qualitative preference queries using SPARQL is expensive [47] while a preference-based physical operator within the OBDA engine that is aware of the properties of preference relations significantly improves the query performance. Firstly, we have defined QualQT, an algorithm that considers a preference-based physical operator integrated into an OBDA engine, and our results suggest that QualQT achieves a comparable performance to this operator implemented within an RDB engine for most cases.

Secondly, under a closed environment characterized by just one concurrent query execution, we have empirically shown that current state-of-the-art tools do not scale adequately for large volumes of data and QualQT significantly outperforms the state-of-the-art methods for larger datasets. Very little attention has been paid to the challenges associated with implementing preferences as a physical operator within a triplestore. We have implemented our technique QualQT on EXASol database and we have demonstrated that qualitative preference queries can benefit significantly from such an implementation. State-of-the-art tools conceive of the preference-based operator as the top-most operator in a query plan, however, a preference-based operator can be pushed down, decreasing the cardinality of the intermediate results, causing important performance benefits and motivating the need for a physical operator.

Finally, a preference-based operator as a physical operator in an OBDA engine also provides the user the ability to express qualitative preference queries where the preference criteria are not the main operator, as in a subquery. State-of-the-art tools may not be able to resolve a query where preferences are expressed in a subquery if they are developed to compute preferences as the top most operator. In this case, the database engine must process the preferences on the subquery before evaluating the outer query since the preferences operator is not integrated in the engine.

## 7. Conclusions and future work

In this article, we have described Morph-Skyline++, an OBDA-based engine that identifies a subset of data in a virtual knowledge graph that meets the criteria of a user request expressed as a SPARQL qualitative preference query. Unlike previous works, we have included the DIFF directive for skyline queries. This directive supports grouping the skyline set by the attribute that comes before the directive. We have also designed and implemented QualQT, an algorithm based on a SPARQL-to-SQL query translation able to compute the preferred set of virtual knowledge graphs. We studied and analyzed the performance of QualQT with respect to the state-of-the-art methods. We reported experimental results with SPREFQL and the skyTPF- and brTPF-based methods, which are state-of-the-art tools to evaluate preference queries over RDF data. These results show that our approach is able to precisely compute the preferences independently of the size of the datasets or the number of dimensions, and it outperforms state-of-the-art tools for larger datasets where just one concurrent query is executed in a closed environment. These tools are not able to produce complete answers in some cases. Through measures such as precision, recall, and F-measure, different answers were obtained when running some preference queries by the state-of-the-art tools. If we further consider that the data are already materialized in RDF, SPREFQL, and the skyTPF- and brTPF-based methods will be better than QualQT when the preferred set is small w.r.t. dataset size.

In the future, we plan to incorporate qualitative preferences into federated queries over SPARQL endpoints extending the approaches already proposed by Endris and colleagues [21], and Mami and colleagues [37]. We also want to evaluate qualitative SPARQL queries over data sources in other data formats such as CSV or JSON, by extending the idea of enforcing implicit constraints exploiting declarative mapping rules presented in Morph-CSV [9]. In addition, with the results presented in this paper, we devise a research line focused on the development of physical operators in triplestores to enhance the performance of qualitative SPARQL queries over materialized RDF knowledge graphs. Also, since TPF-oriented approaches put primary emphasis on low server load and guaranteeing server availability for a large number of concurrent clients, we want to analyze the scalability in terms of the number of concurrent clients for SPARQL qualitative preference query execution, for which the TPF-based method may very well outperform our approach. Finally, our approach can serve as a strong baseline/benchmark for further efforts towards supporting preferences in the native RDF/SPARQL setting considering what has been achieved already for relational databases.

## Acknowledgements

This research is financially supported by Ministerio de Ciencia e Innovación, Spain, under grant Knowledge Spaces (PID2020-118274RB-I00) and by an FPI grant (BES-2017-082511).

## References

[1] R. Agrawal and E.L. Wimmers, A framework for expressing and combining preferences, in: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, Dallas, Texas, USA, May 16–18, 2000, W. Chen, J.F. Naughton and P.A. Bernstein, eds, ACM, 2000, pp. 297–306. doi:10.1145/342009.335423.

[2] ARQ – A SPARQL Processor for Jena, Accessed: 2019-11-08.

[3] I. Bartolini, P. Ciaccia and M. Patella, Efficient sort-based skyline evaluation, *ACM Trans. Database Syst.* **33**(4) (2008), 31:1–31:49. doi:10.1145/1412331.1412343.

[4] J.L. Bentley, H.-T. Kung, M. Schkolnick and C.D. Thompson, On the average number of maxima in a set of vectors and applications, *Journal of the ACM (JACM)* **25**(4) (1978), 536–543. doi:10.1145/322092.322095.

[5] T. Berners-Lee, J. Hendler and O. Lassila, The semantic web, *Scientific american* **284**(5) (2001), 34–43. doi:10.1038/scientificamerican0501-34.

[6] S. Börzsönyi, D. Kossmann and K. Stocker, The skyline operator, in: *Proceedings of the 17th International Conference on Data Engineering*, Heidelberg, Germany, April 2–6, 2001, D. Georgakopoulos and A. Buchmann, eds, IEEE Computer Society, 2001, pp. 421–430. doi:10.1109/ICDE.2001.914855.

[7] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro and G. Xiao, Ontop: Answering SPARQL queries over relational databases, *Semantic Web* **8**(3) (2017), 471–487. doi:10.3233/SW-160217.

[8] S. Chaudhuri, N. Dalvi and R. Kaushik, Robust cardinality and cost estimation for skyline operator, in: *Data Engineering, ICDE'06. Proceedings of the 22nd International Conference on*, Los Alamitos, CA, USA, IEEE Computer Society, 2006, p. 64. ISBN 0-7695-2570-9. doi:10.1109/ICDE.2006.131.

[9] D. Chaves-Fraga, E. Ruckhaus, F. Priyatna, M. Vidal and Ó. Corcho, Enhancing Virtual Ontology Based Access over Tabular Data with Morph-CSV, *Semantic Web* (2021).

[10] A. Chebotko, S. Lu and F. Fotouhi, Semantics preserving SPARQL-to-SQL translation, *Data Knowl. Eng.* **68**(10) (2009), 973–1000. doi:10.1016/j.datak.2009.04.001.

[11] J. Chomicki, Querying with intrinsic preferences, in: *Advances in Database Technology – EDBT 2002*, Springer, Berlin, Heidelberg, 2002, pp. 34–51. ISBN 978-3-540-45876-0. doi:10.1007/3-540-45876-X_5.

[12] J. Chomicki, Preference formulas in relational queries, *ACM Trans. Database Syst.* **28**(4) (2003), 427–466. doi:10.1145/958942.958946.

[13] J. Chomicki, Logical foundations of preference queries, *IEEE Data Eng. Bull.* **34**(2) (2011), 3–10, http://sites.computer.org/debull/A11june/Chomicki1.pdf.

[14] J. Chomicki, P. Godfrey, J. Gryz and D. Liang, Skyline with presorting, in: *ICDE*, U. Dayal, K. Ramamritham and T.M. Vijayaraman, eds, IEEE Computer Society, 2003, pp. 717–719, http://dblp.uni-trier.de/db/conf/icde/icde2003.html#ChomickiGGL03. ISBN 0-7803-7665-X.

[15] E.F. Codd, Relational Completeness of Data Base Sublanguages, Research Report / RJ / IBM /, San Jose, California, 1972, republished on "ACM SIGMOD Anthology".

[16] R. Cyganiak, *A Relational Algebra for SPARQL, Semantic Web*, 2005, Technical Report HPL–2005-170, HP Labs.

[17] S. Das, S. Sundara and R. Cyganiak, *R2RML: RDB to RDF Mapping Language. Working Group Recommendation, World Wide Web Consortium (W3C)*, Sep 2012, 2012, http://www.w3.org/TR/r2rml.

[18] A. Dimou, M.V. Sande, P. Colpaert, R. Verborgh, E. Mannens and R.V. de Walle, RML: A generic language for integrated RDF mappings of heterogeneous data, in: *LDOW, CEUR Workshop Proceedings*, Vol. 1184, CEUR-WS.org, 2014.

[19] D. Dubois, A. Hadjali, H. Prade and F. Touazi, Erratum to: Database preference queries – a possibilistic logic approach with symbolic priorities, *Ann. Math. Artif. Intell.* **73**(3–4) (2015), 359–363. doi:10.1007/s10472-014-9446-2.

[20] M. Endres and E. Glaser, Indexing for skyline computation – a comparison study, in: *Flexible Query Answering Systems – 13th International Conference, FQAS 2019, Proceedings*, Amantea, Italy, July 2–5, 2019, A. Cuzzocrea, S. Greco, H.L. Larsen, D. Saccà, T. Andreasen and H. Christiansen, eds, Lecture Notes in Computer Science, Vol. 11529, Springer, 2019, pp. 31–42. doi:10.1007/978-3-030-27629-4_6.

[21] K.M. Endris, P.D. Rohde, M.-E. Vidal and S. Auer, Ontario: Federated query processing against a semantic data lake, in: *Database and Expert Systems Applications*, Springer International Publishing, Cham, 2019, pp. 379–395. ISBN 978-3-030-27615-7. doi:10.1007/978-3-030-27615-7_29.

[22] P.C. Fishburn, Preference structures and their numerical representations, *Theor. Comput. Sci.* **217**(2) (1999), 359–383. doi:10.1016/S0304-3975(98)00277-1.

[23] P. Godfrey, R. Shipley and J. Gryz, Maximal vector computation in large data sets, in: *VLDB*, ACM, 2005, pp. 229–240, http://dblp.uni-trier.de/db/conf/vldb/vldb2005.html#GodfreySG05. ISBN 1-59593-177-5.

[24] M. Goncalves, D. Chaves-Fraga and O. Corcho, Morph-skyline: Virtual ontology-based data access for skyline queries, in: *2020 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, 2020, pp. 299–307. doi:10.1109/WIIAT50758.2020.00043.

[25] M. Gueroussova, A. Polleres and S.A. McIlraith, SPARQL with qualitative and quantitative preferences, in: *Proceedings of the 2nd International Workshop on Ordering and Reasoning, OrdRing 2013, Co-Located with the 12th International Semantic Web Conference (ISWC 2013)*, Sydney, Australia, October 22nd, 2013, I. Celino, E.D. Valle, M. Krötzsch and S. Schlobach, eds, CEUR Workshop Proceedings, Vol. 1059, CEUR-WS.org, 2013, pp. 2–8, http://ceur-ws.org/Vol-1059/ordring2013-paper1.pdf.

[26] V. Hristidis, N. Koudas and Y. Papakonstantinou, PREFER: A system for the efficient execution of multi-parametric ranked queries, in: *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, Santa Barbara, CA, USA, May 21–24, 2001, S. Mehrotra and T.K. Sellis, eds, ACM, 2001, pp. 259–270. doi:10.1145/375663.375690.

[27] E. Iglesias, S. Jozashoori, D. Chaves-Fraga, D. Collarana and M.-E. Vidal, SDM-RDFizer: An RML interpreter for the efficient creation of RDF knowledge graphs, in: *ACM Intern. Confer. on Information and Knowledge Management*, Vol. **CIKM**, 2020.

[28] C. Kalyvas and T. Tzouramanis, A Survey of Skyline Query Processing, *Computing Research Repository Journal* (2017), arXiv:1704.01788.

[29] I. Keles and K. Hose, Skyline queries over knowledge graphs, in: *The Semantic Web – ISWC 2019*, Springer International Publishing, Cham, 2019, pp. 293–310. ISBN 978-3-030-30793-6. doi:10.1007/978-3-030-30793-6_17.

[30] W. Kießling, Foundations of preferences in database systems, in: *Proceedings of the 28th International Conference on Very Large Data Bases, VLDB'02*, VLDB Endowment, 2002, pp. 311–322. doi:10.1016/B978-155860869-6/50035-4.

[31] W. Kießling, M. Endres and F. Wenzel, The preference SQL system – an overview, *IEEE Data Eng. Bull.* **34**(2) (2011), 11–18, http://sites.computer.org/debull/A11june/Kiebling.pdf.

[32] W. Kießling and G. Köstler, Preference SQL – design, implementation, experiences, in: *Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002*, Hong Kong, August 20–23, 2002, Morgan Kaufmann, -2002, pp. 990–1001. http://www.vldb.org/conf/2002/S30P03.pdf. doi:10.1016/B978-155860869-6/50098-6.

[33] D. Kossmann, F. Ramsak and S. Rost, Shooting stars in the sky: An online algorithm for skyline queries, in: *Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002*, Hong Kong, August 20–23, 2002, Morgan Kaufmann, 2002, pp. 275–286, http://www.vldb.org/conf/2002/S09P01.pdf. doi:10.1016/B978-155860869-6/50032-9.

[34] M. Lacroix and P. Lavency, Preferences; putting more knowledge into queries, in: *VLDB'87, Proceedings of 13th International Conference on Very Large Data Bases*, Brighton, England, September 1–4, 1987, P.M. Stocker, W. Kent and P. Hammersley, eds, Morgan Kaufmann, 1987, pp. 217–225, http://www.vldb.org/conf/1987/P217.PDF.

[35] D. Lanti, G. Xiao and D. Calvanese, VIG: Data scaling for OBDA benchmarks, *Semantic Web* **10**(2) (2019), 413–433. doi:10.3233/SW-180336.

[36] T. Lukasiewicz, M.V. Martinez and G.I. Simari, Preference-based query answering in datalog+/− ontologies, in: *IJCAI 2013*, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3–9, 2013, 2013, pp. 1017–1023.

[37] M.N. Mami, D. Graux, S. Scerri, H. Jabeen, S. Auer and J. Lehmann, Squerall: Virtual ontology-based access to heterogeneous and large data sources, in: *International Semantic Web Conference*, Springer, 2019, pp. 229–245.

[38] S. Mandl, O. Kozachuk, M. Endres and W. Kießling, Preference analytics in EXASolution, in: *Datenbanksysteme für Business, Technologie und Web (BTW)*, LNI, Vol. P-241, GI, 2015, pp. 613–632, https://dl.gi.de/20.500.12116/2434.

[39] F. Michel, J. Montagnat and C. Faron Zucker, A survey of RDB to RDF translation approaches and tools, Research Report, I3S, 2014, ISRN I3S/RR 2013-04-FR 24.

[40] P.F. Patel-Schneider and D. Martin, EXISTStential aspects of SPARQL, in: *Proceedings of the ISWC 2016 Posters & Demonstrations Track Co-Located with 15th International Semantic Web Conference (ISWC 2016)*, Kobe, Japan, October 19, 2016, T. Kawamura and H. Paulheim, eds, CEUR Workshop Proceedings, Vols 1690, CEUR-WS.org, 2016, http://ceur-ws.org/Vol-1690/paper72.pdf.

[41] P.F. Patel-Schneider, A. Polleres and D. Martin, Comparative preferences in SPARQL, in: *Knowledge Engineering and Knowledge Management*, Springer International Publishing, Cham, 2018, pp. 289–305. ISBN 978-3-030-03667-6. doi:10.1007/978-3-030-03667-6_19.

[42] A. Poggi, D. Lembo, D. Calvanese, G.D. Giacomo, M. Lenzerini and R. Rosati, Linking data to ontologies, *J. Data Semantics* **10** (2008), 133–173. doi:10.1007/978-3-540-77688-8_5.

[43] F. Priyatna, Ó. Corcho and J.F. Sequeda, Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph, in: *23rd International World Wide Web Conference, WWW'14*, Seoul, Republic of Korea, April 7–11, 2014, ACM, 2014, pp. 479–490. doi:10.1145/2566486.2567981.

[44] W. Siberski, J.Z. Pan and U. Thaden, Querying the semantic web with preferences, in: *The Semantic Web – ISWC 2006*, Springer, Berlin, Heidelberg, 2006, pp. 612–624. ISBN 978-3-540-49055-5. doi:10.1007/11926078_44.

[45] U. Straccia, On the top-k retrieval problem for ontology-based access to databases, in: *Flexible Approaches in Data, Information and Knowledge Management*, Springer International Publishing, Cham, 2014, pp. 95–114. ISBN 978-3-319-00954-4. doi:10.1007/978-3-319-00954-4_5.

[46] R.E. Tarjan, Efficiency of a good but not linear set union algorithm, *J. ACM* **22**(2) (1975), 215–225. doi:10.1145/321879.321884.

[47] A. Troumpoukis, S. Konstantopoulos and A. Charalambidis, An extension of SPARQL for expressing qualitative preferences, in: *The Semantic Web – ISWC 2017*, Springer International Publishing, Cham, 2017, pp. 711–727. ISBN 978-3-319-68288-4. doi:10.1007/978-3-319-68288-4_42.