SPARQL Query Result Explanation for Linked Data

Rakebul Hasan[1], Kemele M. Endris[1,2], and Fabien Gandon[1]

[1] INRIA Sophia Antipolis, Wimmics, France
[2] DISI, University of Trento, Italy
hasan.rakebul@inria.fr,keme686@gmail.com,fabien.gandon@inria.fr

Abstract. In this paper, we present an approach to explain SPARQL query results for Linked Data using *why-provenance*. We present a non-annotation-based algorithm to generate *why-provenance* and show its feasibility for Linked Data. We present an explanation-aware federated query processor prototype and show the presentation of our explanations. We present a user study to evaluate the impacts of our explanations. Our study shows that our query result explanations are helpful for end users to understand the result derivations and make trust judgments on the results.

1 Introduction

As a result of the W3C Linked Open Data Initiative, recently we have seen a rapid growth in publishing data sets on the Semantic Web, in form of RDF data with SPARQL query endpoints. This enables developers to query and integrate disparate Semantic Web data. As argued in [14, 16], it is essential to provide additional explanations about which source data were used in providing results, how the source data were combined, to enable users understand the result derivations, and validate or invalidate the results.

Within the Semantic Web community, explanations have been studied for Semantic Web applications and OWL entailments. Explanation for SPARQL query results has not been independently studied by the community. However, there have been several works on tracing the origin of query results – e.g. *why-provenance*. These attempts are based on the annotation approach (the eager approach) where the underlying data model, the query language, and the query processing engine are re-engineered to compute provenance during the query processing. This is undesirable for the Linked Data scenario as re-engineering the underlying data model, the query language, or the query processor is often not possible from the querying side. Furthermore, previous work on explanations for the Semantic Web does not study how explanations impact the end-users.

To address these problems, we provide SPARQL query result explanations. The main component in an explanation for a query result tuple is its *why-provenance*. We propose a non-annotation approach to generate *why-provenance* for SPARQL query results. We present an explanation-aware federated query

processor prototype to show the presentation of our explanations. Finally, we present a user study which evaluates the impacts of SPARQL query result explanations on the end-users.

The structure of the rest of this paper is as follows: in section 2, we present the related work. In section 3, we discuss SPARQL query result explanations, introduce the concept of *why-provenance*, and present our algorithm to generate *why-provenance*. In section 5, we present our explanation-aware federated query processor prototype. In section 6, we present a user study to evaluate the impacts of explanations. Finally, we conclude and discuss the future work in section 7.

2 Related Work

Previous work on explanation in the Semantic Web literature [7] addresses the problems of representing explanation metadata [13], and generating explanations for Semantic Web applications [10] and entailments [8]. SPARQL query result explanation has not be studied in the previous work. Query result provenance has been studied in the database community [2] and the Semantic Web community. The previous works on provenance for SPARQL query results are based on transforming the RDF data model and SPARQL query language to relational data model and relational database query language respectively [14, 4], or generation of provenance metadata during the query processing [16, 3]. However, in the Linked Data scenario, we do not have any control over the underlying data model or the query processor. Therefore, re-engineering the underlying data model or query processor is often not possible in the Linked Data scenario. Furthermore, the impacts of explanations on end-users has not been studied in the previous work on explanation in the Semantic Web literature. In the other fields, Lim *et al.* [9] studied the impacts of explanations on end-users for context-aware applications. Tintarev and Masthoff [15] studied the effectiveness of explanations for recommender systems.

3 Explaining SPARQL Query Results

We provide SPARQL query result provenance as query result explanations. More precisely, for a SPARQL query result tuple, we provide its *why-provenance* as its explanation. Buneman *et al.* [1] first introduced the notion of *why-provenance* for relational databases. *Why-provenance* captures all the different witnesses for a tuple in the query result. For a query Q and output tuple t, a *witness* is the sufficient subset of the database records which ensures that the tuple t is in the output. Each witness is a derivation for the output tuple. Theoharis *et al.* [14] later adapted *why-provenance* for RDF and SPARQL. Similar to the relational setting, *why-provenance* for RDF and SPARQL captures all the different derivations of a tuple in the query result. To illustrate, we use a simple example, containing RDF data about professors and the courses they teach, shown in Figure 1. We use identifiers for each triple for presentation purpose in this paper. Consider the SPARQL query *Q1* shown in Listing 1.1, which asks for all the

	Triples about professors
t1	:ProfA :dept :CS
t2	:ProfA :name "Prof. A"
t3	:ProfA :email "a@email.edu"
t4	:ProfA :course :CS101
t5	:ProfA :course :CS103
t6	:ProfA :course :CS201
t7	:ProfA :course :CS204
t8	:ProfB :dept :MATH
t9	:ProfB :name "Prof. B"
t10	:ProfB :email "b@email.edu"
t11	:ProfB :course :MATH101
t12	:ProfB :course :MATH201

	Triples about courses
t13	:CS101 :courseType :underGrad
t14	:CS103 :courseType :underGrad
t15	:MATH101 :courseType :underGrad
t16	:CS201 :courseType :grad
t17	:CS204 :courseType :grad
t18	:MATH201 :courseType :grad

Fig. 1. Example RDF triples.

professors who teach undergraduate level courses and their corresponding email addresses. The first triple pattern *?course :courseType :underGrad* in the query *Q1* selects the undergraduate level courses.

Listing 1.1. SPARQL query Q1

```
SELECT DISTINCT ?name ?email
WHERE
  { ?course :courseType :underGrad .
    ?prof :course ?course .
    ?prof :email ?email .
    ?prof :name ?name
  }
```

Result of Q1:

?name	?email
Prof. A	a@email.edu
Prof. B	b@email.edu

The second triple pattern *?prof :course ?course* selects the professors for those undergraduate level courses. The next two triple patterns *?prof :email ?email* and *?prof :name ?name* selects the email addresses and names of the corresponding professors matched by the two previous triple patterns. The result of the query *Q1* (under set semantics) executed on the RDF data containing the triples in Figure 1 is shown on the right in Listing 1.1. The *why-provenance* for the result tuple (Prof. A, a@email.edu) is {{t14, t5, t2, t3}, {t13, t4, t2, t3}}. Each inner set in *why-provenance* represents a derivation involving the triples in the inner set. This means that the result tuple (Prof. A, a@email.edu) can be derived in two different ways according to *Q1*. The first one by using the triples t14, t5, t2, and t3. The second one by using the triples t13, t4, t2, and t3. The *why-provenance* for the result tuple (Prof. B, b@email.edu) on the other hand has one derivation: {{t15, t11, t10, t9}}. Please not that we are using the triple identifiers only for presentation purpose. The original data model containing the triples shown in Figure 1 is not changed – *i.e.* we do not annotate the RDF triples. We use the RDF triples as they are in the original data source.

3.1 Algorithm for Generating Why-Provenance

In this section, we present our non-annotation approach to generate *why-provenance* for SPARQL query results. We currently do not support SPARQL queries with

sub-queries, FILTER (NOT) EXISTS, MINUS, property paths, and aggregates. The *GenerateWhyProvenace* procedure shown in Algorithm 1 generates *why-provenance* for an RDF model M, a SPARQL query Q, and a result tuple t. The RDF model M can be an RDF dataset or a SPARQL endpoint on which the SPARQL query Q is solved and the result tuple t is produced. At line 2

Algorithm 1 Why-provenance algorithm.

```
1: procedure GenerateWhyProvenace(M,Q,t)
2:     Q' ← ProvenanceQuery(Q,t)
3:     I ← Q'(M)
4:     E ← AlgebricExpression(Q)
5:     W ← DerivationsFromQuery(M,E,I)
6:     return W
7: end procedure
```

of Algorithm 1, we first re-write the original query to a provenance query by adding the tuple t as a solution binding using the SPARQL 1.1 VALUES construct, and projecting all the variables. The result set of the provenance query provides us all the variable bindings on the RDF data for the solution tuple t. Each tuple (row) in the result set of the provenance query represent a derivation for the solution tuple t. The main idea behind our algorithm is to extract *why-provenance* triples from the triple patterns in the original query by replacing the variables in the triple patterns by the corresponding values from each tuple (row) of result of the provenance query. At line 3 of Algorithm 1, we execute the re-written query. At line 4, we convert the original SPARQL query Q to SPARQL algebraic expression for ease of query parsing and manipulation. At line 5, the *DerivationsFromQuery* procedure extracts the derivations. Algorithm 2 shows the *ProvenanceQuery* procedure to re-write the original query to a provenance query. Line 2 adds the result tuple t as a solution binding using the SPARQL 1.1 VALUES construct. Line 3 modifies the query to projects all the variables in the query.

Algorithm 2 Procedure for creating the provenance query.

```
1: procedure ProvenanceQuery(Q,t)
2:     Q' ← AddValueBindings(Q',t)
3:     Q'' ← ProjectAllVariables(Q')
4:     return Q''
5: end procedure
```

Algorithm 3 shows the *DerivationsFromQuery* procedure to extract the derivations given the RDF model M, the SPARQL algebraic expression E, and the provenance query results I. Lines 3–20 iterate through all the tuples of I, extracts provenance triples corresponding to each tuple, and stores them in a set of a sets D. We assume that basic a graph pattern in a SPARQL query is not repeated. We use a hash table, BP, to flag which basic graph pattern (BGP) is examined for a tuple in I to extract provenance triples. Lines 4–6 initialize the hash table by setting *False* for each BPG, meaning none of the basic graph

Algorithm 3 Procedure for extracting derivations from a query.

```
 1: procedure DerivationsFromQuery(M,E,I)
 2:     D ← ∅
 3:     for each tuple in I do
 4:         for each bgp in E do
 5:             BP[bgp] ← False
 6:         end for
 7:         T ← ∅
 8:         if hasUnion(E) or hasJoin(E) or hasLeftJoin(E) then
 9:             for each operator in E do
10:                 T1 ← TriplesForOperator(M, operator, tuple, BP)
11:                 if T1 ≠ ∅ then
12:                     T ← T ∪ T1
13:                 end if
14:             end for
15:         else
16:             bgp ← GetTheBGP(E)
17:             T ← TriplesFromBGP(M, bgp, tuple, BP)
18:         end if
19:         D ← D ∪ {T}
20:     end for
21:     return D
22: end procedure
```

patterns is examined for the current tuple in I at this point. If a query has just one BGP, we extract the provenance triples from that BGP (lines 15–18) for a tuple in I and store the provenance triples in set T. If a query has more than one BGP, *i.e.* if the algebraic expression has the union or the join or the left-join operator, we extract the provenance triples from the operand BGPs of each of the operators and store the provenance triples in set T (lines 7–14) for a tuple in I. We only extract provenance triples for a BGP once at this stage – using the hash table BP as flags for BGPs to keep trace of which BGP has been used so far to extract provenance triples. Finally line 19 does a union of the triples extracted for a tuple in I, stored in set T, as an element (shown by braces around T at line 19) with the set of sets D and assigns the result of the union to D. When we go out of the loop started at line 3, D contains all the derivations we extracted. We return the set of sets D at line 21. Each element in D is a set representing a derivation for the result tuple. Algorithm 4 shows the *TriplesForOperator* procedure which extracts provenance triples from the operands of an operator. Lines 3–4 get the left and the right BGPs for the operator Op. As we are only restricted to SPARQL queries without sub-queries, the operands are always BGPs. Lines 5–7 extract provenance triples from the left BGP L if provenance triples have not been extracted from L yet, and assigns them to the set P. Lines 8–11 extract provenance triples from the right BGP R, stored in the set T, if provenance triples have not been extracted from R yet, and assigns the union of P and T to P. At line 12, we return the set P which

Algorithm 4 Procedure for extracting triples from operands of an operator.

```
1: procedure TriplesForOperator(M,Op,Tup,BP)
2:     P ← ∅
3:     L ← GetLeftBGP(Op)
4:     R ← GetRightBGP(Op)
5:     if BP[L] = False then
6:         P ← TriplesFromBGP(M, L, Tup, BP)
7:     end if
8:     if BP[R] = False then
9:         T ← TriplesFromBGP(M, R, Tup, BP)
10:        P ← P ∪ T
11:    end if
12:    return P
13: end procedure
```

contains all the provenance triples extracted from the left and the right BGPs of the operator *Op*. The *TriplesFromBGP* procedure calls at line 6 and line 8 check if all the triples extracted from the BGPs exist in the RDF model M by sending SPARQL ASK queries with each extracted triples. This means that a BGP which was an operand of a SPARQL UNION or OPTIONAL operator would contribute to the provenance triples only if it matches against the RDF model M. Algorithm 5 shows the *TriplesFromBGP* procedure which does this. Lines

Algorithm 5 Procedure for extracting triples from a basic graph patter.

```
1: procedure TriplesFromBGP(M,BGP,Tup,BP)
2:     T ← ∅
3:     for each triplePattern in BGP do
4:         triple ← ReplaceVariablesByValues(triplePattern, Tup)
5:         if Ask(M, triple) = True then
6:             T ← T ∪ triple
7:         else
8:             BP[BGP] ← True
9:             return ∅
10:        end if
11:    end for
12:    BP[BGP] ← True
13:    return T
14: end procedure
```

3–11 iterate through the triple patterns in the BGP and extracts the triples. At line 4 we replace the variables of a triple pattern by the corresponding values in the tuple *Tup*, where *Tup* is a tuple from the result of the re-written provenance query. Lines 5–6 first check if the extracted triple is valid by sending an ASK query with this triple to the RDF model M, then if it's a valid triple we take the

triple and store it in the set T. If the triple is not valid (does not exist in M), we set the flag for the BGP to true and return an empty set (lines 7–9). At line 10, we go out of the loop started at line 3, and set the flag for the BGP to true. Finally at line 11 we return the set of extracted provenance triples.

4 Performance Evaluation of the Algorithm

We implement our algorithm using Jena-ARQ API[3]. We evaluate our algorithm using the DBPSB benchmark [11] queries on a Jena-TDB (version 1.0.0) triple store [12]. DBPSB includes 25 query templates which cover most commonly used SPARQL query features in the queries sent to DBpedia[4]. We generate our benchmark queries from these query templates. We allow Jena-TDB to use 16 GB of memory. We execute all the queries in a commodity server machine with a 4 core Intel Xeon 2.53 GHz CPU, 48 GB system RAM, and Linux 2.6.32 operating system. As the RDF dataset, we use the DBpedia 3.5.1 dataset with 100% scaling factor – provided by the DBPSB benchmark framework. To generate benchmark queries, we assign randomly selected RDF terms from the RDF dataset to the placeholders in the DBPSB query templates. We generate 1 query for each template resulting total 25 queries. Before executing the queries, we restart the triple store to clear the caches. Then we execute the 25 queries and along with the *why-provenance* algorithm for all the result tuples once in our warm-up phase. Then we execute each query and the *why-provenance* algorithm for all the result tuples of each query 5 times. We report the average execution time and average provenance generation time for all result tuples (PGT) for each query, both in milliseconds (ms). We specify a 300 second timeout for a query execution. Queries belonging to templates 2, 16, 20, and 21 did not finish executing within the 300 seconds time limit, and hence we do not report them.

4.1 Query Execution and Provenance Generation

Table 1 shows the number for results (#RES), query executing time (QET), provenance generation time for all result tuples (PGT), and provenance generation time per result tuple (PGTPR) for DBPSB queries. PGTs for queries with long execution times and large number of results (queries 6, 8, 10, 14, 22, 24, and 25) are very high. This is not surprising because for each result tuple of a query, we execute the original query with the result tuple as a variable-value binding. Database literature already discusses this issue [2]. Generally speaking, non-annotation approaches compute provenance only when it is needed, by examining the source data and the output data. This requires sophisticated computations involving the source data and the output data. This means each individual tuple in the output data has to be examined separately to compute their provenance, and hence time required for generating provenance for all the

[3] http://jena.apache.org/
[4] http://dbpedia.org

result tuples for a query is high. However, in contrast to the annotation approaches (as in [16]), our approach does not affect the query execution time. In addition, our goal is to provide provenance as query result explanations. We only need provenance for the result tuple for which the explanation is asked. Therefore, provenance generation time per result tuple (PGTPR) is the interesting measure for us. PGTPR for all the queries are low, ranging from 0.001

Query	#RES	QET (ms)	PGT (ms)	PGTPR (ms)
1	4	25	12.2	3.05
3	1	75	65.6	65.6
4	2	8495.6	8.4	4.2
5	13	78	102.6	7.89
6	3238	785	428.2	0.13
7	21	4.2	57.8	2.75
8	60447	7392.4	1035.4	0.017
9	4	1156.2	341.2	85.3
10	2933	6506.8	164828	56.2
11	1	0.4	0.01	0.01
12	1	18.4	43.8	43.8
13	2	0.4	0.4	0.2
14	4137	604.6	7999.6	1.93
15	38	925.6	0.2	0.005
17	82	20.6	0.6	0.007
18	34	0.6	0.2	0.006
19	2	0.4	0.002	0.001
22	82298	7424.4	405456.4	4.927
23	1	16.6	17.8	17.8
24	134968	5729	1700.4	0.013
25	47696	1683.4	1036758.2	21.737

Table 1. Query execution and provenance generation times for DBPSB queries.

ms to 85.8 ms. Even for the long running queries, PGTPR values are low. This is because we add the variable-value binding to the original query to compute provenance, which makes the query simpler to solve for the query processor.

5 An Explanation-Aware Federated Query Processor

We developed a prototype system for federated query processing with explanation features. Users can ask for explanation for each query result tuple in our system. We implement a virtual integration-based federated query processor. The first step for our federated query processing is selecting the data sources by sending SPARQL ASK queries with each triple pattern. Next, we split the original query to sub-queries, sequentially send them to the relevant data sources (nested loop join), and combine the result in the local federator. Each sub-query is a CONSTRUCT SPARQL query which returns a set of matched triples for its triple patterns. We create a local virtual graph combining the resulted triples from all the sub-queries, then locally solve the original query on this virtual

graph using Jena-ARQ. We borrow the idea of CONSTRUCT sub-queries from Corese-DQP [5]. We also implement the common federated query processing concepts of exclusive triple pattern groups and bound join proposed in [**?**].

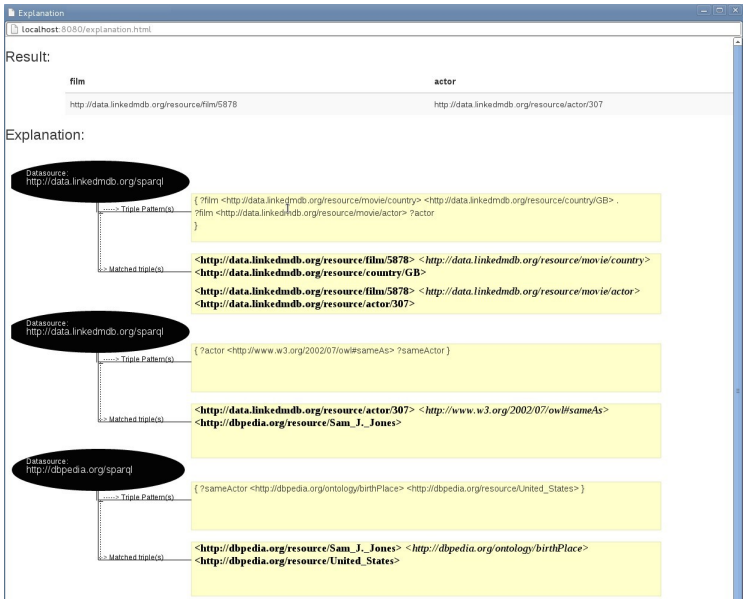


Fig. 2. Example of a query result explanation.

We provide a user interface to enable users to configure SPARQL endpoints as data sources, and submit queries. Furthermore, users can ask for explanation for each query result tuple from the user interface. We provide three types of information in an explanation. We show the *why-provenance* triples, which data source each triple in the *why-provenance* comes from, and which triple pattern of the original query each triple in the *why-provenance* matches. Figure 2 shows an example of a query result explanation. We generate the *why-provenance* triples using the algorithm we presented in section 3.1 on the local virtual RDF graph. We keep two additional indexes in the federated query processor to keep trace of which data source each triple comes from, and which triple pattern each triple matches. These two indexes allow us to provide the information on data sources and matched triple patterns in the explanations.

6 Evaluation of the Impacts of Explanations

We conducted a user study to investigate the impact of query result explanations. Our study is similar to the user study conducted by Lim *et al.* [9] to examine effectiveness of different types of explanations for context-aware intelligent systems. The questionnaire for our study consists of three sections: learning section, reasoning section, and survey section. Furthermore, we have two cases:

with explanation and without explanation. A participant is randomly assigned to the case of "with explanation" or "without explanation".

In the learning section, participants were given a high-level overview of our query processor and an example SPARQL query with a result tuple to help them learn how the federated query processor works. Participants for the "with explanation" case additionally received the explanation of the result tuple for the example query (as shown in Figure 2). In the reasoning section, participants were given the same SPARQL query as in the learning section, but a different result tuple along with the some triples contained in two data sources (DBpedia[5] and LinkedMDB[6]). Then we first ask the participants to select the relevant data sources for each triple pattern in the query. Next, we ask the participants to select the source triples (*why-provenance* triples) from the two data sources which contributed to the result tuple. Then we ask the participants to rate their confidence on their answer choices for the data source selection and the source triple selection questions. The choices for confidence rating were very low, low, medium, high, and very high. The questions in the reasoning section help us analyze how the users understand the result derivation process and if the explanation provided in the learning section have any impact on their understanding. In the survey section of our study, we ask the participants if explanations help users to understand the result derivation and to make trust judgments on the results. Furthermore, we ask them which types of information they think are helpful in an explanation for understanding and making trust judgments. The questions in the survey section help us understand how the participants feel about the system and its explanation features.

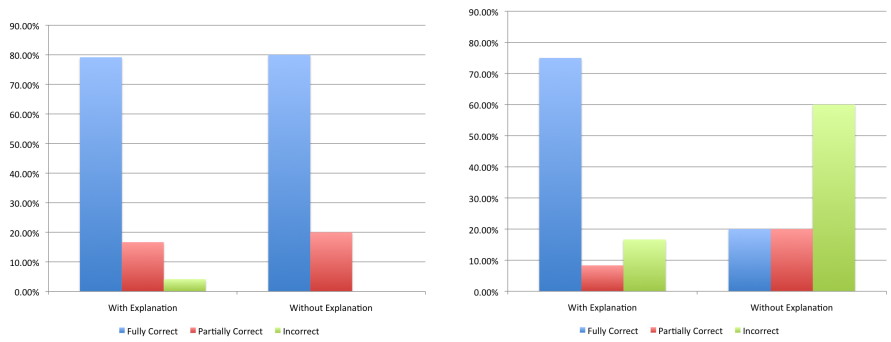
The query we used is a query to find the British movies with American actors. The result tuple includes URIs for a film and an actor. Part of the query is solved in LinkedMDB (finding the British movies) and part of it is solved in DBpedia (finding birth places of the actors). In the query result tuple, we intentionally do not provide natural language descriptions. Instead we provide URIs from LinkedMDB – which are numeric resource URIs – for the actor and the film. This is to make sure that participants are not using their background knowledge about movies and actors in their answers. For the data source selection and source triple selection questions, we provide small subsets of DBpedia triples (11 triples) and LinkedMDB triples (13 triples). We used Google Forms[7] for the questionnaires and Google App Engine[8] to randomize the selection of two cases – "with explanation" or "without explanation". We invited the member of our laboratory[9] (via our mailing list), the members of Semantic Web Interest Group[10] (via their mailing list), and the followers of Twitter hashtags #SemanticWeb, #RDF, and #SPARQL. 11 participants took part in the study. There

[5] `http://dbpedia.org/`
[6] `http://linkedmdb.org/`
[7] `http://www.google.com/google-d-s/createforms.html`
[8] `https://appengine.google.com/`
[9] `http://wimmics.inria.fr/`, `https://glc.i3s.unice.fr/`
[10] `http://www.w3.org/2001/sw/interest/`

were 6 participants for the "with explanation" case and 5 participants for the "without explanation" case. There were 8 male participants and 3 female participants. The ages of the participants range from 22 to 65. All the participants had knowledge of RDF and SPARQL. The questionnaire and the responses of the participants are available online[11].

6.1 Results of the Study

We analyze the ability of the participants to apply their understanding of the system by computing the number of fully correct, partially correct, and incorrect answers for the data source selection and the source triple selection questions in the reasoning section. If a participant selects all the correct choices for an answer, we consider it as fully correct. If a participant selects all the correct choices but also selects some extraneous choices, we consider the answer as partially correct. If a participant's choices for an answer do not contain all the correct choices, we consider it as incorrect. In addition, if a participant selected all choices given for the source triple selection question, we consider the answer as incorrect to avoid guessing. For the data source selection question, we had 4 questions for 4 triple patterns in the query. We count the number of participants who provided fully correct answers, partially correct answers, and incorrect answers for each of these 4 questions. Then we take the average of the counts for the fully correct answers, the average of the counts for the partially correct answers, and the average of the counts for the incorrect answers. These averages represent the average number of participants into the three answer categories categories – fully correct, partially correct, and incorrect – for the data source selection question as a whole. We compute these averages separately for both the "with explanation" and "without explanation" cases and compute the percentages of participants in the three answer categories for the two cases from these average. Figure 3(a) shows the percentage of participants with fully correct, partially



(a) Data source selection (b) Source triple selection

Fig. 3. Participants' response about data source selection and source triple selection.

[11] `http://ns.inria.fr/ratio4ta/sqe/`

correct, and incorrect answers when the explanation is given and when the explanation is not given for the data source selection question. The results are very similar for both "with explanation" and "without explanation" cases. Majority of the participants understood how data source selection works for our federated query processor system when the explanation was given ((79.17%) and also when the explanation was not given (80.0%). Therefore the impact of explanations for source selection understanding is not clear from our study. For the source triple selection question, we had two questions for the two data sources we used. We compute the percentages of participants in the fully correct, partially correct, and incorrect answer categories for the "with explanation" and "without explanation" cases using the same method as the data source selection question. Figure 3(b) shows the percentage of participants with fully correct, partially correct, and incorrect answers when the explanation is given and when the explanation is not given for the source triple selection question. More participants provided correct answers when the explanation was give (75% for "with explanation", 20% for "without explanation"). Furthermore, more participants provide incorrect answers when the explanation was not given (16.67% for "with explanation", 60% for "without explanation"). This clearly shows that participants who were given explanations understood better which triples contributed to the result from the two data sources. The final question in the reasoning section asks participants to rate their confidence level about the answers for the data source selection question and the source triple selection question. Figure 4 shows the confidence level of the participants about their answers. 50.0% of participants with explanation rate their confidence as very high whereas none of participants without explanation rate very high. 33.33% of participants with explanation rate their confidence as high whereas 80% of participants without explanation rate high. This shows that participants with explanation are more confident on their answers – as many of them answered "very high" or "high".

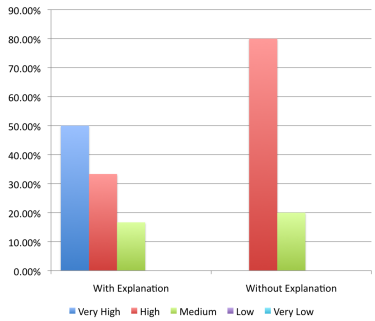
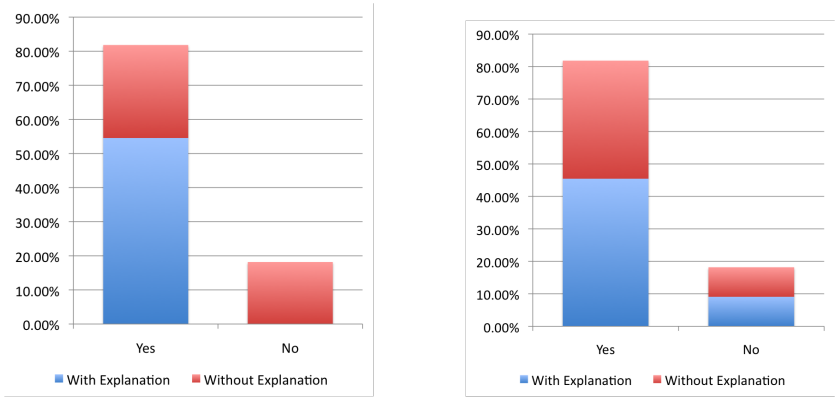


Fig. 4. Participants' confidence level about their answers.

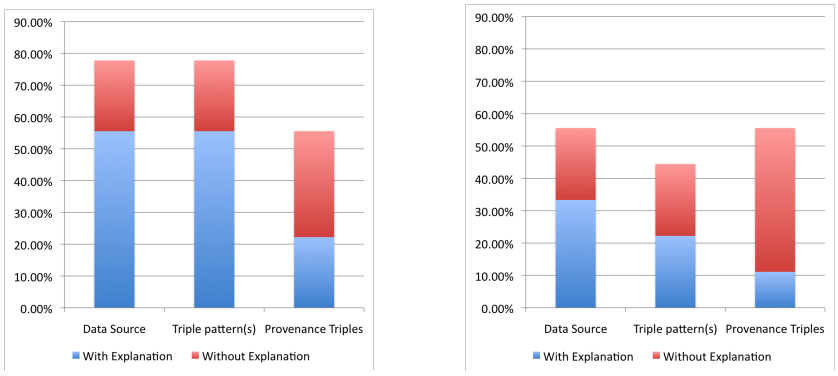
For the survey section, we ask the participants if explanations are helpful to understand the query result derivation, and if explanations are helpful to make trust judgments on the query result. If a participant answered "yes", he/she was also asked what kind of information he/she found helpful. Figure 5(a) shows the percentage of participants who answered explanations are helpful or unhelpful for understanding the query result derivation. Majority of the partic-



(a) Understanding (b) Making trust judgments

Fig. 5. Percentage of participants who answered explanations are helpful or unhelpful.

ipants (81.81) responded that explanations are helpful for understanding the query result derivation. Only 18.18% of the participants answered that explanations are unhelpful for understanding the query result derivation – none of these participants were given explanations. Figure 5(b) shows the percentage of participants who answered explanations are helpful or unhelpful to make trust judgments on the query result. Again, Majority of the participants (total 81.81%) responded that explanations are helpful to make trust judgments on the query result. Only 18.18% of the participants answered that explanations are unhelpful to make trust judgments on the query result. This shows that majority of the survey participants feel that explanations are helpful to understand query result derivations and to make trust judgments on query results. Figure 6(a)



(a) Understanding (b) Making trust judgments

Fig. 6. Participants who found different types of information in the explanation helpful.

shows the participants who found information on data source, triple pattern(s), and *why-provenance* triples helpful for understanding the query result derivation. Please note that only the answers from participants who answered "yes"

shown in Figure 5(a) are considered. Out of 9 participants who answered "yes", 77.78% responded that the data source related information was helpful, 77.78% responded that the triple pattern(s) related information was helpful, and 55.55% responded that the provenance triple related information was helpful. However, our analysis on source selection question responses (Figure 3(b)) shows that the explanation helped participants significantly improve their correctness on selecting the provenance triples. Therefore, it is hard to explain why only 22.22% with explanation responded that the provenance triple related information was helpful. One possible reason could be that when they were not given the explanation, they felt the need for explanation with provenance triple (hence 33.33% for without explanation). But when they were given the explanation, they were not aware that the provenance triple related information helped them to have a better understanding. Figure 6(b) shows the participants who found information on data source, triple pattern(s), and *why-provenance* triples helpful to make trust judgments. Again only the answers from participants who answered "yes" shown in Figure 5(b) are considered. Out of 9 participants who answered "yes", 55.55% responded that the data source related information was helpful, 44.44% responded that the triple pattern(s) related information was helpful, and 55.55% responded that the provenance triple related information was helpful. Again, it is interesting to notice that participants who were not given the explanation felt the need for provenance triples related information. This analysis shown in Figure 6 shows that participants found data source and triple pattern(s) related information helpful for understanding the query result derivation, but have less stronger feeling about provenance triples related information for understanding query result derivations. For making trust judgments, participants do not have as strong opinions, but majority of them feel that data source and provenance triple related information are helpful.

7 Conclusion and Future Work

In this paper, we present an approach to explain SPARQL query results for Linked Data. We present a non-annotation approach to generate *why-provenance* – the main component of an explanation – and show its feasibility for common Linked Data queries. We present an explanation-aware federated query processor prototype and show the presentation of our explanations. Finally, our user study to evaluate the impacts of explanations shows that our query result explanations are helpful for end users to understand the result derivations and make trust judgments on the results.

In the future work, we would like to extend our algorithm to generate *how-provenance*, which explain how a result tuple was derived with the details of the operations performed in the derivation. Furthermore, we would like to conduct the user study with more participants. Finally, we would like to represent our explanations in RDF using explanation vocabularies such as *Ratio4TA* [6].

Acknowledgments: This work is supported by the ANR CONTINT program under the Kolflow project (ANR-2010-CORD-021-02).

References

1. Buneman, P., Khanna, S., Tan, W.C.: Why and where: A characterization of data provenance. In: Proceedings of the 8th International Conference on Database Theory. pp. 316–330. ICDT '01, Springer-Verlag, London, UK, UK (2001)
2. Cheney, J., Chiticariu, L., Tan, W.C.: Provenance in databases: Why, how, and where. Found. Trends databases 1(4), 379–474 (Apr 2009)
3. Corby, O., Gaignard, A., Zucker, C., Montagnat, J.: Kgram versatile inference and query engine for the web of linked data. In: Web Intelligence and Intelligent Agent Technology (WI-IAT), 2012 IEEE/WIC/ACM International Conferences on. vol. 1, pp. 121–128 (Dec 2012)
4. Damásio, C.V., Analyti, A., Antoniou, G.: Provenance for sparql queries. In: Proc. of the 11th International Conference on The Semantic Web - Volume Part I. pp. 625–640. ISWC'12, Springer-Verlag, Berlin, Heidelberg (2012)
5. Gaignard, A.: Distributed knowledge sharing and production through collaborative e-Science platforms. Ph.D. thesis, Universit Nice Sophia Antipolis (2013)
6. Hasan, R.: Generating and summarizing explanations for linked data. In: Presutti, V., dAmato, C., Gandon, F., dAquin, M., Staab, S., Tordai, A. (eds.) The Semantic Web: Trends and Challenges, LNCS, vol. 8465, pp. 473–487. Springer (2014)
7. Hasan, R., Gandon, F.: A Brief Review of Explanation in the Semantic Web. Workshop on Explanation-aware Computing (ExaCt 2012), European Conference on Artificial Intelligence (ECAI 2012) (2012)
8. Horridge, M., Parsia, B., Sattler, U.: Laconic and precise justifications in OWL. In: Proc. of the 7th Int'l Conference on the Semantic Web. pp. 323–338. ISWC '08, Springer-Verlag (2008)
9. Lim, B.Y., Dey, A.K., Avrahami, D.: Why and why not explanations improve the intelligibility of context-aware intelligent systems. In: Proc. of the SIGCHI Conference on Human Factors in Computing Systems. pp. 2119–2128. CHI '09, ACM, New York, NY, USA (2009)
10. McGuinness, D., Furtado, V., Pinheiro da Silva, P., Ding, L., Glass, A., Chang, C.: Explaining semantic web applications. In: Semantic Web Engineering in the Knowledge Society (2008)
11. Morsey, M., Lehmann, J., Auer, S., Ngonga Ngomo, A.C.: Dbpedia SPARQL benchmark performance assessment with real queries on real data. In: Aroyo, L., et al. (eds.) The Semantic Web ISWC 2011, LNCS, vol. 7031, pp. 454–469. Springer Berlin Heidelberg (2011)
12. Owens, A., Seaborne, A., Gibbins, N., mc schraefel: Clustered TDB: A clustered triple store for Jena (November 2008)
13. Pinheiro da Silva, P., McGuinness, D., Fikes, R.: A proof markup language for semantic web services. Information Systems 31(4-5), 381–395 (2006)
14. Theoharis, Y., Fundulaki, I., Karvounarakis, G., Christophides, V.: On provenance of queries on semantic web data. IEEE Internet Computing 15(1), 31–39 (Jan 2011)
15. Tintarev, N., Masthoff, J.: Evaluating the effectiveness of explanations for recommender systems. User Modeling and User-Adapted Interaction 22(4-5), 399–439 (Oct 2012)
16. Wylot, M., Cudre-Mauroux, P., Groth, P.: Tripleprov: Efficient processing of lineage queries in a native rdf store. In: Proceedings of the 23rd International Conference on World Wide Web. pp. 455–466. WWW '14 (2014)