A Proposal for Nested Results in SPARQL

Sébastien Ferré

A Proposal for Nested Results in SPARQL*

Sébastien Ferré**

Univ Rennes, CNRS, IRISA
Campus de Beaulieu, 35042 Rennes, France
Email: ferre@irisa.fr

Abstract. Tables are a common form of query results, notably in SPARQL. However, due to the flat structure of tables, all structure from the RDF graph is lost, and this can lead to duplications in the table contents, and difficulties to interpret the results. We propose an extension of SPARQL 1.1 aggregations to get nested results, i.e. tables where cells may contain embedded tables instead of RDF terms, and so recursively.

1 Motivation

The SPARQL query language [8] offers a powerful way to extract and compute information from RDF datasets. For `SELECT` queries, the results are presented in a table. Each column corresponds to a projected variable in the `SELECT` clause, and each row corresponds to a solution, mapping those variables to RDF terms. Such tables are universally understood, and can be read in two directions, by row or by column. They make a good use of the screen space, compared for instance to graph visualizations, and can therefore display a lot of information at once. They can be dynamically filtered and ordered according to each column. Despite those advantages, tables in general and SPARQL results in particular have drawbacks due to the fact that they are in first normal form (1NF), a notion from relational databases that states that table cells only contain atomic values, here RDF terms. Although it sounds like a reasonable constraint, it has negative consequences on the readability of query results as shown in the following example.

Table 1 shows an excerpt of the results of the following SPARQL query on DBpedia, which retrieves films directed by Danny Boyle, along with their music composers and actors, and also the birth year of actors.

```
SELECT ?f ?mc ?a ?y
WHERE { ?f a dbo:Film ; dbo:director dbr:Danny_Boyle ;
                        dbo:musicComposer ?mc ;
                        dbo:starring ?a .
        ?a dbo:birthYear ?y . }
```

It can be observed that the table contains a lot of redundancies. For instance, the birth year of an actor is repeated for each music composer. The number of rows for a film is

* This research is supported by ANR project PEGASE (ANR-16-CE23-0011-08).

Table 1. Flat table of films by D. Boyle with music composer, actor, and birth year

film	music composer	actor	birth year
Slumdog millionaire	A. R. Rahman	Dev Patel	1990
Slumdog millionaire	A. R. Rahman	Freida Pinto	1984
Slumdog millionaire	A. R. Rahman	Anil Kapoor	1959
Sunshine	John Murphy	Cilian Murphy	1976
Sunshine	John Murphy	Chris Evans	1981
Sunshine	John Murphy	Rose Byrne	1979
Sunshine	John Murphy	...	...
Sunshine	Underworld	Cilian Murphy	1976
Sunshine	Underworld	Chris Evans	1981
Sunshine	Underworld	Rose Byrne	1979
Sunshine	Underworld	...	...
...	...	...	...

Table 2. Nested table of films by D. Boyle with a list of music composers, and a list of actors with birth year

film	music composers	actors
Slumdog millionaire	music composer A. R. Rahman	actor / birth year Anil Kapoor 1959 Freida Pinto 1984 Dev Patel 1990 ... ...
Sunshine	music composer John Murphy Underworld	actor / birth year Cilian Murphy 1976 Rose Byrne 1979 Chris Evans 1981 ... ...
...	...	...

equal to its number of music composers times its number of actors. When ordering by birth year, one needs to order first by film so as to keep rows grouped by film.

The key contribution of this paper is to allow table cells to contain smaller tables instead of RDF terms. Those *nested tables* are obtained by allowing variables to map to sets of solution mappings, in addition to RDF terms. Tables can be deeply nested, with tables containing tables that in turn contain tables, and so on. In practice, RDF terms and nested tables are not mixed randomly inside a table. For a given column, either all cells contain an RDF term or all cells contain a nested table. Moreover, all nested tables in a column are defined on the same variables. This means that our nested tables follow a regular schema, although more complex than that of flat tables.

Table 2 is the nested version of Table 1. The main table has a single row per film, and two of its columns, "music composers" and "actors", contain nested tables. The former column contains one-column nested tables that contain the list of music composers of each film. The latter column contains two-columns nested tables that contain the list of actors of each film, along with their birth year. It can be observed that the nested table does not contain redundancies anymore, and that the dependencies between columns

is made explicit. Music composers and actors are dependent on the film, but not on each other. The birth year is dependent on the actor.

2 Related Work

Nested tables were proposed and formalized in relational databases, where nesting is not only used for query results but also for data tables [7]. The motivation was to relax the first normal form (1NF). The authors extend relational algebra with two operators, *nest* and *unnest*, that operate on subsets of columns. In this work, as we start from RDF graphs instead of tables, we only need a mechanism to nest the table of results, there is no need for unnesting. We also need to adapt the nest operation to SPARQL algebra and grammar.

A lot of work have proposed various forms of visualization of SPARQL results (e.g., maps, charts) in order to help their understanding [1]. They are a valuable complement to the tabular view but that they cannot fully replace it in the general case. A JSON-based query language has been proposed to facilitate the exposition of SPARQL results in APIs [6]. In particular it can generate nested tables similar to ours. However, it only covers a fragment of SPARQL graph patterns, and the grouping criteria is limited to a single variable per table. Some extensions of SPARQL have also been proposed. For instance, a new `CLUSTER BY` clause was proposed to group the rows of the table of results into a hierarchical clustering [5]. In contrast, nested tables can be seen as a form of 2D hierarchical clustering because they involve grouping subsets of columns and subsets of rows at the same time.

3 A New Aggregation Construct

Our proposal is to extend the algebra and grammar of SPARQL 1.1 with a new aggregation construct [4] that computes nested tables (i.e. sequences of solution mappings) instead of RDF terms.

In the above example, Table 1 can be seen as the global multiset of solutions, and Table 2 as the expected result after applying aggregations that compute nested tables instead of RDF terms. After grouping by film, the nested tables in column "music composers" are obtained by *projecting* each solution group on variable "music composer", and by *removing duplicates*. Similarly for column "actors" by projecting on "actor" and "birth year", by removing duplicates, and by ordering rows by birth year. To sum up, the new aggregations need at least projecting on a subset of variables, removing duplicates, and ordering results. Those computations correspond to the category of solution modifiers, which are formalized as transformations between sequences of solution modifiers, and hence as table-to-table transformations. In addition to solution modifiers, we allow groupings (clause `GROUP BY`) in the definition of table aggregations, and hence aggregations in the definition of table aggregations. This makes the definition of aggregations recursive, which enables deeply nested tables.

The main impact of the extension is that a variable may now be bound to a nested table in addition to RDF terms. For the sake of simplicity, we consider that those variables can only be used as projection variables, and produce an error when used in other contexts (e.g., in an expression).

We now extend the grammar of SPARQL 1.1 [8] so as to give a concrete syntax to the extended algebra presented above. It simply consists in extending the *Aggregate*

rule with one new production.

$$\textit{Aggregate} ::= \ \ldots \mid \text{'\{'} \quad \textit{SelectClause} \quad \textit{SolutionModifier} \quad \text{'\}'}$$

The ellipsis stands for existing constructs like `COUNT(DISTINCT ?x)`. Symbol *SelectClause* covers projection (`SELECT`) and removal of duplicates (`DISTINCT`, `REDUCED`). Symbol *SolutionModifier* covers ordering (`ORDER BY`), top-k results (`LIMIT`) and offset (`OFFSET`), grouping (`GROUP BY`), and filtering as a solution modifier (`HAVING`). This production is therefore enough to cover all needed features. We add braces to delimit the aggregation construct, and also by analogy with subqueries. Indeed, our table aggregations are syntactically and semantically equivalent to subqueries where the graph pattern (clause `WHERE`) is implicit.

In the example on films, the query in the extended SPARQL that returns the nested table (Table 2) is the following.

```
SELECT ?f  ({SELECT DISTINCT ?mc} AS ?mcs)
           ({SELECT DISTINCT ?a ?y ORDER BY ?y} AS ?as)
WHERE { ?f a dbo:Film ; dbo:director dbr:Danny_Boyle ;
                        dbo:musicComposer ?mc ;
                        dbo:starring ?a .
        ?a dbo:birthYear ?y . }
GROUP BY ?f
```

Clause `GROUP BY ?f` defines the groups of solutions over which the two table aggregations are evaluated. Variable `?mcs` holds the list of distinct music composers for each film, a one-column nested table. Variable `?as` holds the list of distinct actors and their birth year, in increasing order of birth year, a two-column nested table.

Note that nothing forbids to combine *term* aggregations with *table* aggregations. For example, in the above query, one could add the term aggregation `(COUNT(DISTINCT ?a) AS ?na)` in order to add a column giving the number of actors, for each film, in addition to their list. If those lists are too long, they can be bounded in size by adding a `LIMIT` clause in the table aggregations.

To illustrate deeply nested tables and top-k results, we extend the above query to get the lists of spouses of each actor as tables nested into the actor tables. We also add a `LIMIT` to get only the three top actors per film.

```
SELECT ?f  ({SELECT DISTINCT ?mc} AS ?mcs)
           ({SELECT DISTINCT ?a ?y ({SELECT DISTINCT ?sp} AS ?sps)
             ORDER BY ?y
             LIMIT 3} AS ?as)
WHERE { ?f a dbo:Film ; dbo:director dbr:Danny_Boyle ;
                        dbo:musicComposer ?mc ;
                        dbo:starring ?a .
        ?a dbo:birthYear ?y ; dbo:spouse ?sp }
GROUP BY ?f
```

It can be observed in the above queries that the nesting schema is entirely expressed in the main `SELECT` clause (table aggregations can also be used in subqueries). This suggests that nested results can be obtained by post-processing the flat results, without actually extending SPARQL. However, it would require to retrieve more answers than the number of desired answers, because of the redundancies induced by flat results.

In the example, 11 answers in the flat results are used to generate 2 answers in the nested results. The relation between the two numbers is hard to predict, and can be of combinatorial nature. Integrating nested tables into SPARQL enables to control the number of returned results, and creates opportunities for optimization in query evaluation by trying to avoid redundancies altogether.

Another advantage of the SPARQL extension is to extend SPARQL's expressivity with a new class of aggregations that would be otherwise very difficult to express. The most interesting one is the combination of grouping and top-k results, like "the three youngest actors of each film" or "the last two mayors of the three most populated cities of each country".

4 Conclusion

We have proposed an extension of SPARQL 1.1 aggregations that enables nested tables as query results, i.e. tables that can contain tables in their cells, and so recursively. Nested tables improve the readability of results by avoiding redundancies in their contents, and by exhibiting the dependencies and independencies between their columns. The proposed extension is fully backward compatible, and should be relatively easy to implement in existing query engines. Nested results have been integrated into Sparklis [2], a SPARQL query-builder, as a new view on results. This has to be done by post-processing of flat results until implementations of the proposed extension are available.

References

1. Bikakis, N., Sellis, T.: Exploration and visualization in the web of big linked data: A survey of the state of the art. In: Int. Work. Linked Web Data Management (LWDM) (2016)
2. Ferré, S.: Sparklis: An expressive query builder for SPARQL endpoints with guidance in natural language. Semantic Web: Interoperability, Usability, Applicability 8(3), 405–418 (2017), `http://www.irisa.fr/LIS/ferre/sparklis/`
3. Hoefler, P., Granitzer, M., Sabol, V., Lindstaedt, S.: Linked data query wizard: A tabular interface for the semantic web. In: The Semantic Web: ESWC 2013 Satellite Events, pp. 173–177. Springer (2013)
4. Kaminski, M., Kostylev, E.V., Cuenca Grau, B.: Semantics and expressive power of subqueries and aggregates in SPARQL 1.1. In: Int. Conf. World Wide Web. pp. 227–238. ACM (2016)
5. Ławrynowicz, A.: Query results clustering by extending SPARQL with CLUSTER BY. In: OTM Confederated Int. Conf. on the Move to Meaningful Internet Systems. pp. 826–835. Springer (2009)
6. Lisena, P., Meroño-Peñuela, A., Kuhn, T., Troncy, R.: Easy web API development with SPARQL transformer. In: Int. Semantic Web Conf. pp. 454–470. Springer (2019)
7. Roth, M.A., Korth, H.F., Silberschatz, A.: Extended algebra and calculus for nested relational databases. ACM Transactions on Database Systems (TODS) 13(4), 389–417 (1988)
8. SPARQL 1.1 query language (2012), `http://www.w3.org/TR/sparql11-query/`, w3C Recommendation