# Implementing Joins in SPARQL over RDF Resources

Student Name: Christian Johnston, ID: mdsw22

## I. ABSTRACT

The aim of this project is to implement a number of different join algorithms in SPARQL over RDF resources. We implement inner, left-outer, left-exclusive and full-outer join algorithm in SPARQL, querying the dbpedia RDF endpoint. For each of these joins, we explain the mechanism used as well as to give some justification as to why it is used and how it is useful. We compare the different join approaches by considering the results obtained and how they compare, the performance of the queries in terms of wall-clock running times and how it scales when more documents are joined. We also consider joining RDF graphs from multiple remote endpoints using SPARQL 1.1 Federated Queries. We found that the inner join was simple in terms of its implementation however its efficiency was heavily dependent on the order of the triple patterns in the SPARQL query. Experiments verified the theory that all of the joins scaled approximately linearly with the number of triple patterns in the query. We found the feature of optional pattern matching (left-outer join) to be crucial in Semantic Web applications where every application has only partial knowledge about the resources being managed. Despite this, we found OPTIONAL queries to not be associative or commutative leading to more advanced optimisation implementations. We found the results obtained using the different joins was dependent on both the selectivity of the triple patterns and the completeness of the data sources. We found Federated Queries to be an essential yet inefficient part of the SPARQL query language however the inefficiency could be mitigated through the use of the GRAPH and NAMED GRAPH keywords to narrow the search space.

## II. INTRODUCTION

The goal of the Semantic Web to provide a common framework that allows data to be shared, reused and linked. Uniform Resource Identifiers (URIs) are globally unique identifiers for things whilst the Resource Description Framework (RDF) is the standard for representing semantically linked data. SPARQL is a declarative query language for performing data manipulation and data definition operations on data represented as a collection of RDF triples. The SPARQL query language is based on matching graph patterns. The simplest graph pattern is the triple pattern, an RDF triple where at least one of subject, predicate or object is a variable. Combining triple patterns gives a basic graph pattern, where an exact match to a graph is needed to fulfil a pattern. The purpose of this project is to understand how a query language for semi-structured data can be mapped onto concepts in relational query languages such as SQL. This joining of data is essential for the Semantic Web vision- how databases across the world can be queries and processed together in a semantically coherent way. The project implements and analyses the following joins: **Inner join, Left-outer join**, **Full outer join** and **Left-exclusive join.** RDF is semi-structured data and adherence of this data to its Ontology specification and in turn its completeness, is not always enforced which makes OPTIONAL queries a crucial tool. In fact, Atre claims that the occurrence of OPTIONAL queries tends to be as high as 50% of the total queries in the dbpedia query logs (2013). Inner joins are an essential part of any query language as it provides the ability to determine the data that is consistent between two databases. Full outer join provides useful information about redundancy as well as missing data from both sides and is helpful when determining the whole result set, not just matches. We also consider SPARQL 1.1 Federated Queries which are used to query data from multiple remote endpoints which is essential for the Semantic Web vision of joining data from diverse and remote databases in a semantically coherent way. To compare the join algorithms, we conduct a thorough experimental and analytical comparison by considering a range of evaluation criteria including: difficulty of implementation, difference in results obtained, complexity of join algorithm, run-time on test data and how it scales when more documents are joined, how optimised is the join algorithm in terms of query optimisation and comparison to similar queries in SQL. In terms of tools used, SPARQLWrapper, a SPARQL endpoint interface for Python is used to aid application development, allowing the wall-clock running time of the query to be measured whilst the query results can be displayed in a number of formats including JSON and XLM.

## III. METHODOLOGY

### 1. Inner join

Achieved through Basic Graph Patterns (BGP) which are formed by joining one or more triple patterns. The join is achieved via the period (.) (**A.B) -** Join the results of solving A and B (where A and B are graph patterns) by matching the values of common variables, the entire query pattern must match for there to be a solution.

**Figure 1**: SPARQL inner join example

```
SELECT ?empName ?managName
 WHERE { ?emp      emplP:lastName   ?empName .
         ?emp      emplP:manager    ?manager .
         ?manager  emplP:lastName   ?managName }
```

**2. Left-outer join**

Left-outer joins allow information to be added to the solution where the information is available, but not to have the solution rejected because some part of the query pattern does not match. This is achieved in SPARQL through the OPTIONAL operator applied to a graph pattern: **A OPTIONAL {B, C, …, N}.** This joins together the results of solving A and B, C, …, N by matching the values of common variables. Keep all solutions from A whether or not there is a matching solution in B, C, …N. If an OPTIONAL pattern fails to match a particular solution, any variables in that pattern remain unbound for that solution.

**Figure 2**: SPARQL left-outer join via OPTIONAL operator.

```
PREFIX space: <http://purl.org/net/schemas/space/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?name ?country
WHERE {

  #This pattern must be bound
  ?thing  rdfs:label  ?name .

  #Anything in this block doesn't have to be bound
  OPTIONAL {
    ?thing  space:country  ?country .
  }
}
```

**3. Full-outer join**

Full-outer join is achieved through the UNION operator: **A UNION {B, C, …, N}.** When patterns are combined using the UNION keyword, the resulting combined pattern will match if any of the sub-patterns are matched. One of several alternative graph patterns may match and if more than one of the alternatives matches, all the possible pattern solutions are found. Unlike the OPTIONAL pattern, if neither part of the UNION pattern matches, then the graph pattern will not match.

**Figure 3**: SPARQL full-outer join via UNION operator.

```
PREFIX space: <http://purl.org/net/schemas/space/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?subject ?displayLabel
WHERE {

   {
     ?subject  foaf:name  ?displayLabel .
   }
   UNION
   {
     ?subject  rdfs:label  ?displayLabel .
   }
}
```

**4. Left-exclusive join**

Left-exclusive join achieved via MINUS keyword (subtracted graph patterns) which represents a form of negation within SPARQL. **A MINUS {B}.** Include only those results from A that are not compatible with any of the results from B.

**5. Federated Queries**

SERVICE keyword instructs a query processor to invoke different portions of a SPARQL query against different **remote** SPARQL endpoints in a single query. Results are returned and are combined with results from the rest of the query. Queries may explicitly allow failed SERVICE requests with the use of the SILENT keyword.

**Figure 4**: Federated query SPARQL structure via SERVICE operator.

```
SELECT * WHERE {
   ?localitem ?localproperty ?shareditem .        Local pattern
   SERVICE <http://example.remote.sparql> {
        ?shareditem ?remoteproperty ?remoteitem .  Remote pattern
   }
}
```

**6. Named Graphs**

The FROM keyword lets us specify the target graph from which the data is queried whilst FROM NAMED assigns a name to the input graph. We hypothesize that using this feature in a systematic form may reduce the search space when evaluating federated SPARQL queries- Chaves-Fraga et al determine improvements of up to 10%. (2017)

## IV. Results

For the inner-join, with the inclusion of each additional triple pattern, the solution set gets smaller as the entire query pattern must match for there to be a solution. As such, graph pattern expressions given by the conjunction of triple patterns can be evaluated in $O$ ($|p| \cdot |d|$) where $|p|$ is number of triple patterns in the query and |d| is the number of RDF tuples in the dataset. As $|d| \gg |p|$ in our queries, the join algorithm scales linearly with the number of triple patterns (|p|), as demonstrated in Graph 1. BGP queries are associative and commutative, i.e., a change in the order of triple patterns and joins between them does not change the query result but the efficiency does change- for each triple in a graph pattern, the algorithm loops on the next pattern and retrieves the matching triples. As such, for optimisation purposes, we construct each query with the most restrictive triple patterns first (as per the order in Figure 5), as suggested by Tsialiamanis (2012) , eliminating as many solutions as possible early in the algorithm execution.

$$(s,p,o) \prec (s,?,o) \prec (?,p,o) \prec (s,p,?) \prec$$
$$\prec (?,?,o) \prec (s,?,?) \prec (?,p,?) \prec (?,?,?)$$

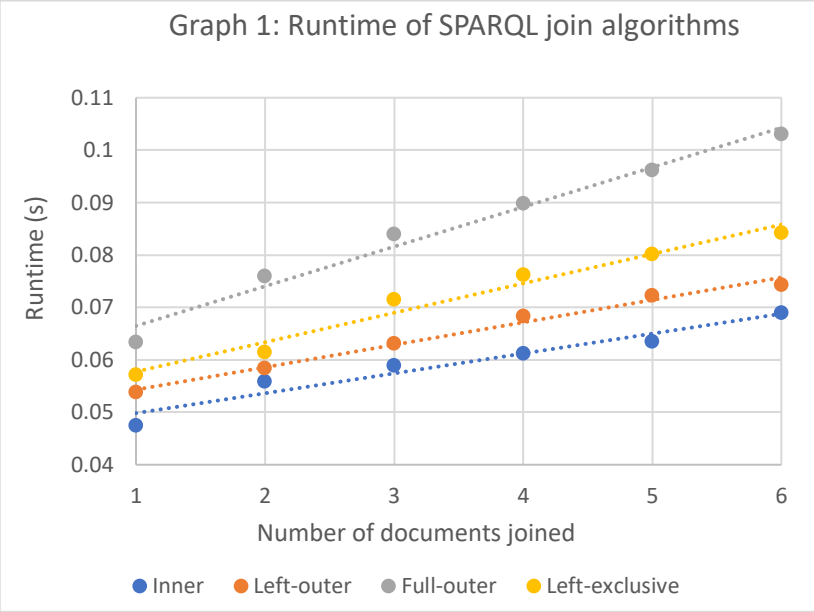
**Figure 5**: Ordering of triple patterns from most, i.e. one that is likely to produce less intermediary results, to least selective.

In SQL, a NULL in any join attribute causes a tuple combination to be rejected whereas in SPARQL, we found that the combination is rejected only if there is a conflict (an attribute is bound on both sides, but to different values). If a shared attribute is unbound on one side, then the value from the other side is taken for the result tuple and if a shared attribute is unbound on both sides, then the attribute is unbound in the result tuple.

**Figure 6**

```
SELECT ?friend ?sitcom WHERE {
  :Jerry :hasFriend ?friend .
  OPTIONAL {
    ?friend :actedIn ?sitcom .
    ?sitcom :location :NewYorkCity .}}
```

It was found that the left-outer join operation via the OPTIONAL operator is not associative or commutative. In Figure 6, if we label each triple pattern as tp1, tp2 and tp3 respectively, the left-outer join between tp1 and tp2 cannot happen before the inner join between tp2 and tp3. As such, the computational time of a left-outer join query is highly dependent on the selectivity *(*fraction of triples in an RDF dataset that contain a bound subject, predicate or object in a sub-pattern*)* of the triple patterns in the inner-join. In terms of run-time, we found the left-outer join to scale linearly with the number of joins as shown in Graph 1. The impact a triple pattern in the OPTIONAL clause has on the result data is highly dependent on how complete the corresponding data is in the RDF store. If the data in the joining RDF stores is fully complete and consistent, the OPTIONAL operator behaves similarly to the disjunction (inner-join) operator. This is reflected in the results in Graph 1 whereby the two algorithms scale at the same rate with the left-outer join taking longer due to partially incomplete joining data and thus the large solution set.




For full-outer join queries, we find that UNION is both commutative and associative and it scales similarly to the inner-join: O (p) as shown in Graph 1. One point to note is that the wall-clock running time or the full-outer join is higher than the inner and left-outer joins, due to the fact that, in comparison to the inner join and the left-outer join, the size of the solution-set is not narrowed with the addition of each join.

The runtime performance of a SPARQL query has a positive correlation to the number of RDF triples it is evaluated against and that query performance is inversely proportional to the number of common variables across separate named graphs. As such, efficient queries seek to localise SPARQL sub-patterns by using the NAMED GRAPH operator to specify the subset of triples in a dataset that portions of the query should be evaluated against, reducing the search space and thus increasing efficiency. The performance of SPARQL federated queries was found to be particularly poor due to the communication overheads of sending results between remote end-points.

## V. Conclusion

In conclusion, we found each of the joins to scale approximately linearly with the number of triple patterns however the efficiency of the inner join was heavily dependent on the order of the triple patterns in the SPARQL query. We found the feature of optional pattern matching to be crucial in semantic Web applications where it is assumed that every application has only partial knowledge about the resources being managed. Despite this, we found OPTIONAL queries to not be associative or commutative leading to more advanced optimisation implementations. If I had more time, I would attempt to implement some of the query optimisation algorithms discussed in a number of papers as the efficiency of the join query algorithms is highly dependent on the order of the triple patterns. For example, Atre has suggested mitigating the issues caused by the restrictions on the reorderability of left-outer-joins by representing a nested BGP-OPTIONAL query with a graph of supernodes (Atre, 2013). Another avenue of further work would be to apply optimisation heuristics such as the position of joins within the queries as in (Petros Tsialiamanis, 2012) or to quantify the efficiency of Federated Queries and how it scales when the number of remote endpoints is increased.

**References**

**Foundations of SPARQL Query Optimization** [Journal] / auth. Michael Schmidt Michael Meier, Georg Lausen // Database Theory. - [s.l.] : ICDT, 2010. - Vol. 1. - pp. 4-33.
**Heuristics-based Query Optimisation for SPARQL** [Journal] / auth. Petros Tsialiamanis Lefteris Sidirourgos, Irini Fundulaki, Vassilis Christophides, Peter Boncz // Extending Database Technology. - [s.l.] : ACM, 2012. - Vol. 1. - pp. 324-335.
**Left Bit Right: For SPARQL Join Queries with OPTIONAL Patterns (Left-outer-joins)** [Journal] / auth. Atre Medha. - [s.l.] : cs.ox.ac.uk, 2013. - Vol. 1.
**On the Formulation of Performant SPARQL Queries** [Journal] / auth. Antonis Loizou Paul Groth, Renzo Angles // Journal of Web Semantics. - [s.l.] : Creative Commons license, 2015. - Vol. 21. - pp. 1-26.
**On the Role of the GRAPH Clause in the Performance of Federated SPARQL Queries** [Journal] / auth. David Chaves-Fraga Claudio Gutierrez, and Oscar Corcho // Dataset Profiling and Federated Search for Web Data. - 2017. - Vol. 1.

**Annex**

We query the **dbpedia** RDF store via the dbpedia SPARQL endpoint *http://dbpedia.org/sparql*).

1. **Table of data for Graph 1** (all values in seconds, average over 1000 iterations)

| Number of joins | Inner | Left-outer | Full-outer | Left-exclusive |
|---|---|---|---|---|
| **1** | 0.0475 | 0.05376 | 0.0634 | 0.05714 |
| **2** | 0.0558 | 0.058366 | 0.0759 | 0.06143 |
| **3** | 0.0589 | 0.063057 | 0.084 | 0.0715 |
| **4** | 0.06125 | 0.068255 | 0.0898 | 0.0762 |
| **5** | 0.0635 | 0.0723 | 0.0961 | 0.0802 |
| **6** | 0.068955 | 0.07432 | 0.1030 | 0.0842 |

2. **Code for timing the SPARQL queries**

```
# average the time over 1000 repeated trials
totalTime = 0
for i in range(0, 1000):

    start = time.time()

    # fire query
    results = sparql.query()

    end = time.time()
    print(end - start)
    totalTime += (end - start)

finalAvg = totalTime/1000
print finalAvg
```

### 3. SPARQL inner join via Basic Graph Pattern

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
SELECT distinct ?rugbyplayer ?nationalteam ?numcaps ?countryOfBirth ?population ?birthdate

WHERE
{
    ?rugbyplayer a dbo:RugbyPlayer.
    ?rugbyplayer dbp:ruPosition <http://dbpedia.org/resource/Centre_(rugby_union)>.
    ?rugbyplayer dbo:birthPlace ?countryOfBirth.
    ?rugbyplayer dbp:ruNationalteam ?nationalteam.
    ?rugbyplayer dbp:ruNationalcaps ?numcaps.
    ?nationalteam dbp:worldCupApps ?numworldcupappearances.
    ?rugbyplayer dbo:birthDate|dbp:birthDate ?birthdate.

}
```

### 4. SPARQL left-outer join OPTIONAL operator

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
SELECT distinct ?rugbyplayer
WHERE
{

    ?rugbyplayer dbo:birthDate|dbp:birthDate ?birthdate

    OPTIONAL
    {
        ?rugbyplayer dbp:ruPosition <http://dbpedia.org/resource/Centre_(rugby_union)>
    }
    OPTIONAL
    {
        ?rugbyplayer dbo:birthPlace ?countryOfBirth
    }
    OPTIONAL
    {
       ?rugbyplayer dbp:ruNationalteam ?nationalteam
    }
    OPTIONAL
    {
       ?nationalteam dbp:worldCupApps ?numworldcupappearances
    }
    OPTIONAL
    {
        ?rugbyplayer a dbo:RugbyPlayer
    }
    OPTIONAL
    {
        ?rugbyplayer dbp:ruNationalcaps ?numcaps.
    }
    OPTIONAL
    {
        ?rugbyplayer dbo:birthDate|dbp:birthDate ?birthdate.
    }
}
```

**5. SPARQL full-outer join UNION operator**

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
SELECT distinct ?rugbyplayer ?nationalteam ?numcaps ?countryOfBirth
WHERE
{
    {
        ?rugbyplayer dbo:birthDate|dbp:birthDate ?birthdate
    }

    UNION
    {
        ?rugbyplayer dbp:ruPosition <http://dbpedia.org/resource/Centre_(rugby_union)>
    }

    UNION
    {
        ?rugbyplayer dbo:birthPlace ?countryOfBirth
    }

    UNION
    {
       ?rugbyplayer dbp:ruNationalteam ?nationalteam
    }

    UNION
    {
        ?rugbyplayer dbp:ruNationalcaps ?numcaps
    }
    UNION
    {
       ?nationalteam dbp:worldCupApps ?numworldcupappearances
    }
}
```

**6. SPARQL full-outer join UNION operator**

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
SELECT distinct ?rugbyplayer ?nationalteam ?numcaps ?countryOfBirth
WHERE
{
    {
        ?rugbyplayer dbo:birthDate|dbp:birthDate ?birthdate
    }

    MINUS
    {
        ?rugbyplayer dbp:ruPosition <http://dbpedia.org/resource/Centre_(rugby_union)>
    }

    MINUS
    {
        ?rugbyplayer dbo:birthPlace ?countryOfBirth
    }

    MINUS
    {
       ?rugbyplayer dbp:ruNationalteam ?nationalteam
    }
    MINUS
    {
        ?rugbyplayer dbp:ruNationalcaps ?numcaps
    }
}
```

**7. Federated Query**

```
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT ?name
WHERE
{
    SERVICE SILENT <http://dbdpedia.org/sparql>
    {
        SELECT ?name WHERE
        {
            ?pais a dbo:Country.
            ?pais rdfs:label ?name.
        }
    }
}
```