Knowledge Graph Lab

Tejaswini Dhupad

Mohammad Zain Abbas

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

May, 2022

TABLE OF CONTENTS

Our GitHub repository’s link can be found here

B. ONTOLOGY CREATION

B1. TBOX Definition

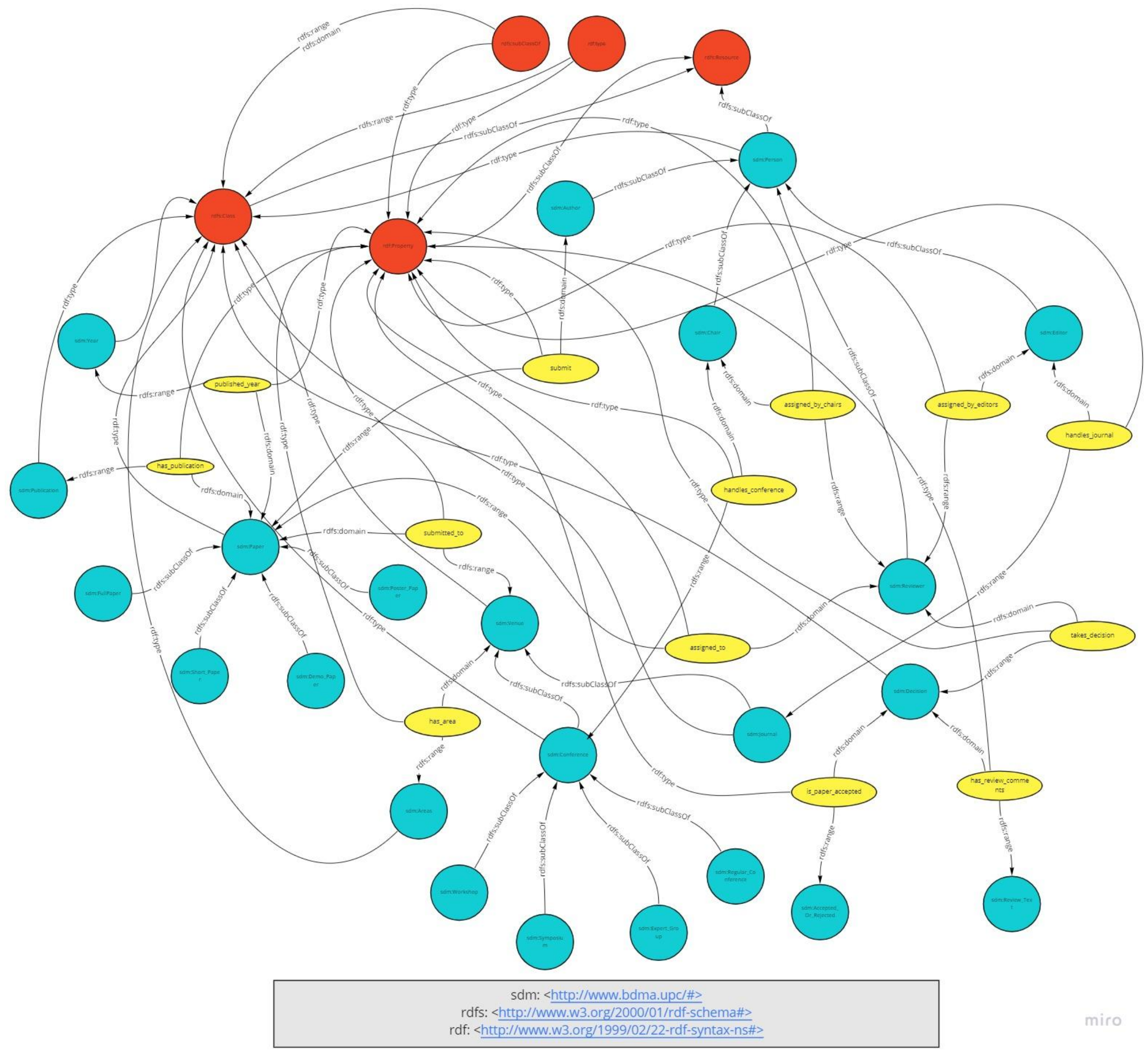


Fig. 1: Graphical representation for TBOX[1]

Note: you can find a clearer graphical representation here.

We created TBOX via Apache Jena's API. First, we created our ontology model with rdfs specification. Then, we added ontology classes and properties for each atomic concept in our TBOX. You can find the TBOX creation code here.

We added "*Person*" as a super-class for "*Author*", "*Chair*", "*Editor*" and "*Reviewer*". Similarly, added "*Full_Paper*", "*Short_Paper*", "*Demo_Paper*" and "*Poster*" as subclasses of "*Paper*". We also created "*Venue*" as a super-class for "*Conference*" and "*Journal*". And then

[1]NOTE: We have just linked the major classes to the *rdfs:Class* via *rdf:type*. In order to minimise the complexity of the TBOX, we haven't done the same with subclasses. All the subclasses are also linked with the *rdfs:Class* via *rdf:type*.

created “*Workshop*”, “*Symposium*”, “*Expert_Group*” and “*Regular_Conference*” as subclasses of “*Conference*”.

We decided to save the final decision (accepted/rejected and reviewer text) by both reviewers as a single instance per decision. Means, for a given paper P, reviewer A and reviewer B will have a final (and common) decision D, based on which we considered our paper as publication (if accepted). As a design choice, we made a dummy class “*Decision*” to hold the common and final decision for both reviewers, and had two sub-classes “*Accepted_Or_Rejected*” and “*Review_Text*”.

When both reviewers make a decision and the paper is accepted, it is published either in Conference Proceedings or in Journal Volume and becomes a publication. We decided to use the already existing *rdfs:Class* “*Conference*” and “*Journal*” to differentiate between conference proceedings and journal volume. And created *rdfs:Class* for “Publication” only.

And for areas, we decided to programmatically create the sub-classes on the fly. This was done when creating ABOX. You can find the code snippet here about how we are creating the subclasses of “*Areas*” on the fly. The main reasons why we decided to do this were: 1) we don’t know how many different areas we have in the data beforehand. 2) to avoid hard-coded areas.

We wrote the TBOX ontology model to a file, in order to use it when creating ABOX.

B2. ABOX Definition

We created ABOX via Apache Jena API. Our basic methodology to create ABOX was quite simple: load the ontology model (TBOX), parse the csv file with instance data. And creating instances of each atomic concept and linking with TBOX will be done automatically (since we used the same ontology model).

First, we loaded the saved ontology model with rdfs specification. We get all the classes and properties defined in the model. Then, we read and parse the csv file containing all the instances for preproceed publications’ data (you can find the guide about dataset and preprocessing on github repo).

After reading and parsing each row, we first clean the string using a helper method. This method removes all special characters, white spaces and some escape characters. After that, we then created an individual for each concept (that were mentioned in our TBOX) using the OntClasses we got from our model, and added properties (whenever needed) to link the nodes with each other.

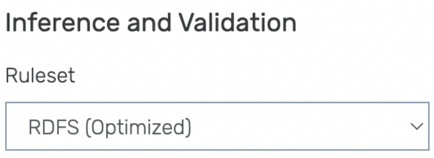
Once all the nodes and properties are done, we outstream the ABOX model as a .nt file with output language as “*N-TRIPLE*”.

You can find the ABOX creation code here.

B3. Create the final ontology

As mentioned in B2, the linking between TBOX and ABOX was done automatically as we used the same ontology model in ABOX creation which was saved after creating TBOX. While creating ABOX, we created some links using addProperty method, and the rest of the links were inferred by Graph-DB's reasoner automatically.

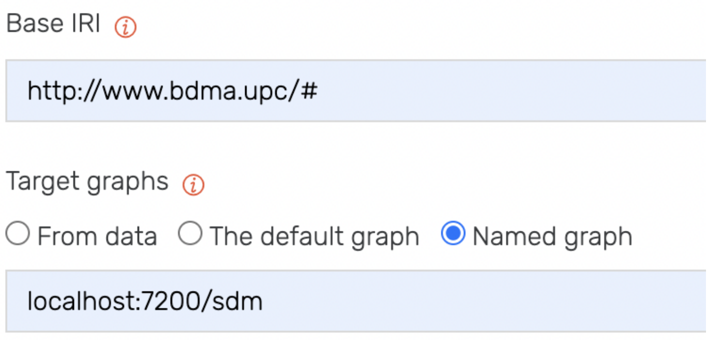
As suggested, we used the "*RDFS (Optimized)*" ruleset for inference (to get the expected inference entailed by *rdfs:domain* and *rdfs:range* in GraphDB) when loading our ontology in GraphDB.


Inference and Validation

Ruleset: RDFS (Optimized)

Inference ruleset

We used the default values for all other options. We imported the data as "Named graph" (same as what we did for dbpedia data).

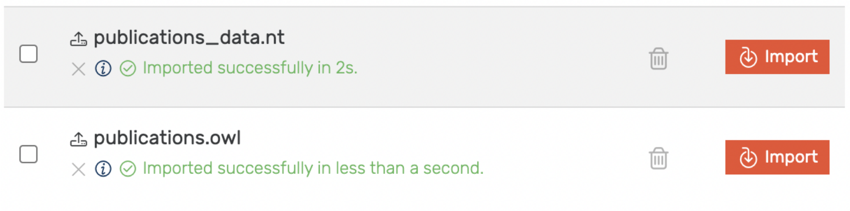

Base IRI: http://www.bdma.upc/#

Target graphs: From data [✓ Named graph] The default graph

localhost:7200/sdm

Import RDF files as Named Graph

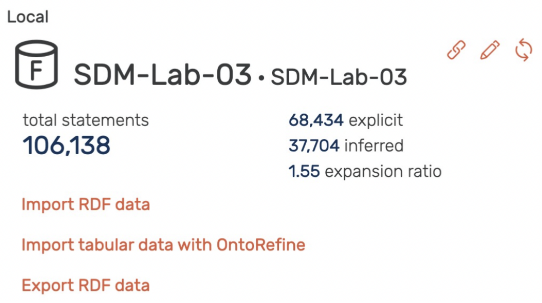
And imported our publications' data to GraphDB


publications_data.nt — Imported successfully in 2s.

publications.owl — Imported successfully in less than a second.

Imported .nt (ABOX) and .owl (TBOX) files in GraphDB

After generating all the statements, more than ⅓ links were inferred automatically, thanks to reasoning.


Local

SDM-Lab-03 • SDM-Lab-03

total statements 106,138

68,434 explicit
37,704 inferred
1.55 expansion ratio

Import RDF data
Import tabular data with OntoRefine
Export RDF data

Basic GraphDB repository statistics

Summary Statistics:

Below is the summary statistics of the graph obtained after linking the ABOX with TBOX:

Measure	Value
Total number of classes	7727
Total number of properties	26
Total number of instances	31,096
Total number of triples	106,138

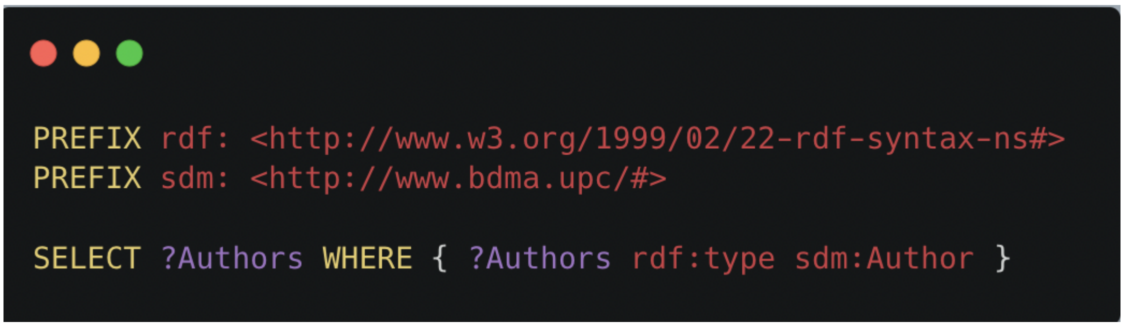
Percentage of number of instances by class :

Class	Percentage of Total Instance
Author	100
Chair	17.28
Editor	17.89
Reviewer	55.01
Full_Paper	30.14
Short_Paper	28.32
Demo_Paper	30.14
Poster_Paper	12.40
Conference	62.80
Journal	65.51

Workshop	31.37
Symposium	35.29
Expert_Group	35.68
Regular_Conference	35.29

B4. Querying the ontology

1. Find all Authors



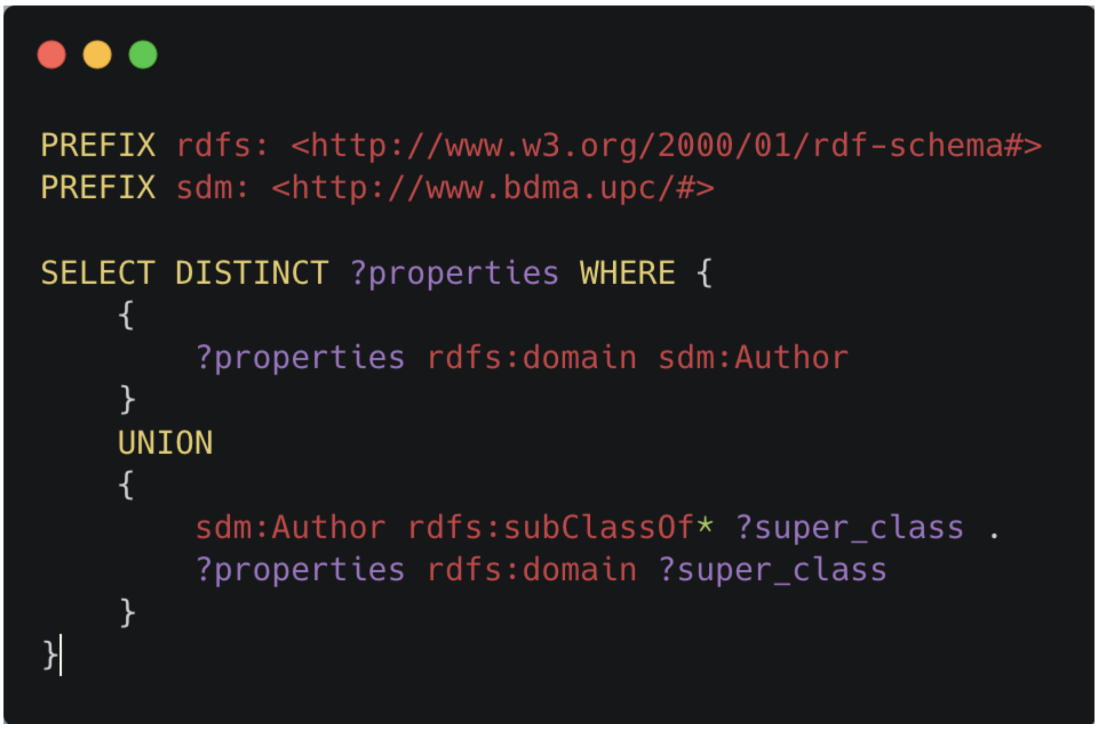
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX sdm: <http://www.bdma.upc/#>

SELECT ?Authors WHERE { ?Authors rdf:type sdm:Author }
```

Fig. 2: SPARQL query for finding all Authors

As mentioned in TBOX and ABOX creation, we have "Author" *rdfs:Class*, so finding all authors was pretty straight forward; just search all data instances which have the *rdf:type* of "Author". You can find the query and respective output in our github repo.

2. Find all properties whose domain is Author



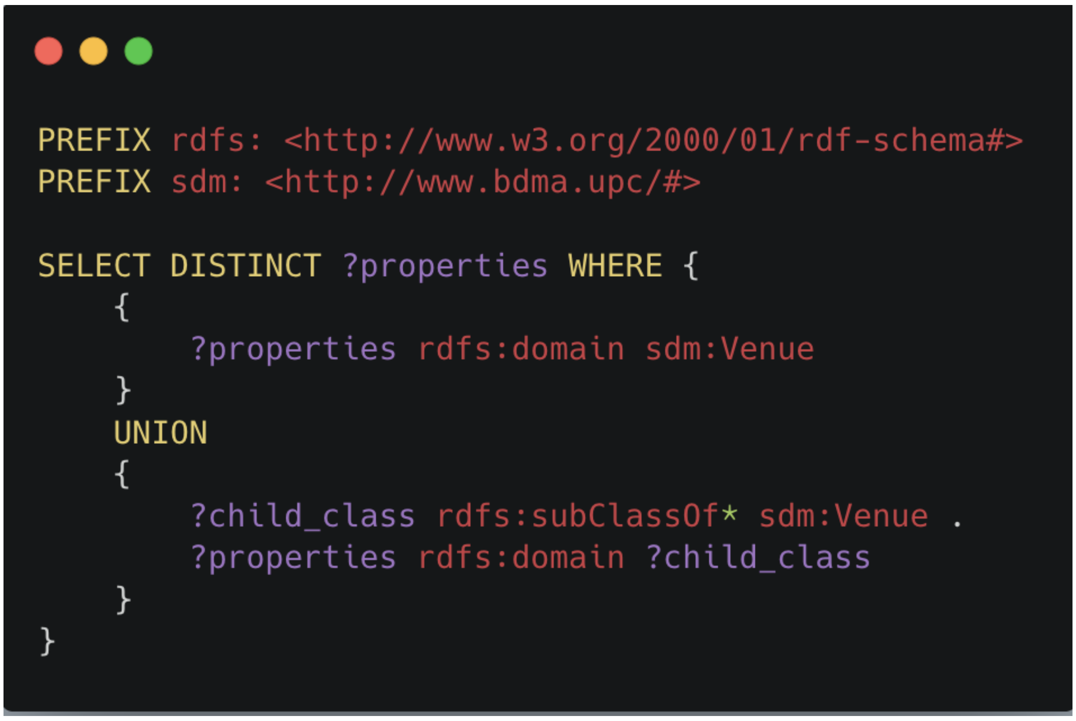
```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sdm: <http://www.bdma.upc/#>

SELECT DISTINCT ?properties WHERE {
    {
        ?properties rdfs:domain sdm:Author
    }
    UNION
    {
        sdm:Author rdfs:subClassOf* ?super_class .
        ?properties rdfs:domain ?super_class
    }
}
```

Fig. 3: SPARQL query for finding all properties with domain Author

Since, properties of super class are inferred automatically for sub-class in *rdfs*, we splitted our query into two parts (to cover all cases). First part is checking all properties whose *rdfs:domain* is “Author”; and the second part is checking properties whose *rdfs:domain* are the super-classes of “Author”. We combine the result and return. You can find the query and respective output in our github repo.

3. Find all properties whose domain is either Conference or Journal



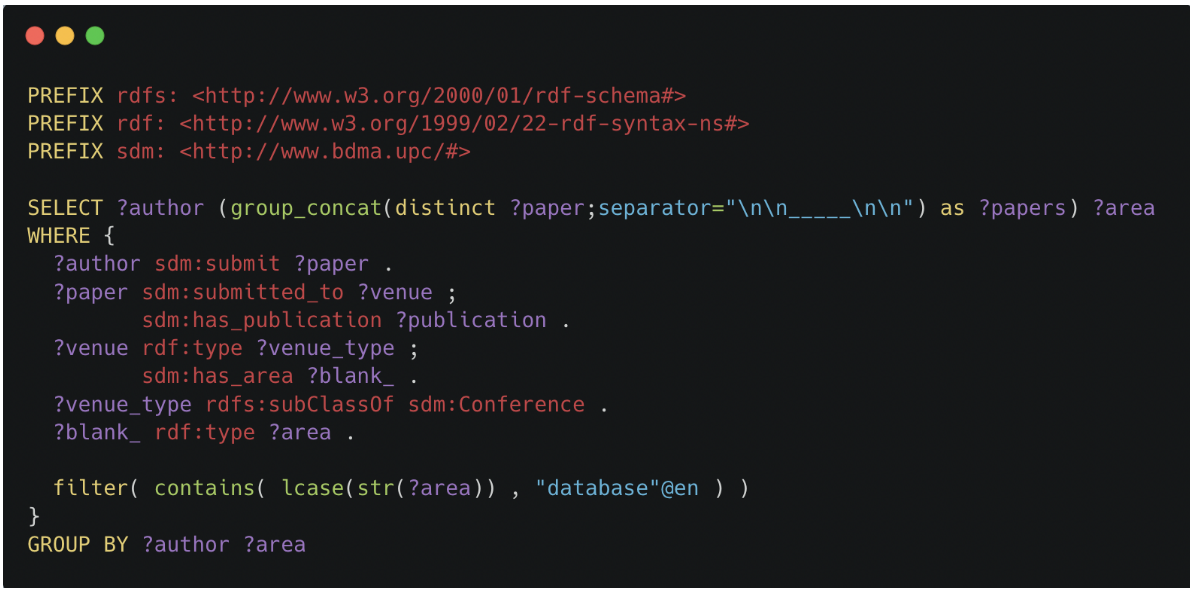
```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sdm: <http://www.bdma.upc/#>

SELECT DISTINCT ?properties WHERE {
    {
        ?properties rdfs:domain sdm:Venue
    }
    UNION
    {
        ?child_class rdfs:subClassOf* sdm:Venue .
        ?properties rdfs:domain ?child_class
    }
}
```

Fig. 4: SPARQL query for finding all properties with domain Conference or Journal

This SPARQL query is very similar to the last one, but here we have multiple *rdfs:Class* to check against. As mentioned in the TBOX, we have sub-classes for “Venue” and then further sub-classes for “Conference”. Keeping them in mind, we followed the same idea that we applied in the last query, and first got all the properties with “Venue” as their domain. And then later, we got all properties with subclasses of “Venue” (‘*’ ensures that we are getting all the subclasses of “Venue”) as their domain. We combine the result and return. You can find the query and respective output in our github repo.

4. Find all the papers written by a given author that where published in database conferences



```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX sdm: <http://www.bdma.upc/#>

SELECT ?author (group_concat(distinct ?paper;separator="\n\n_____\n\n") as ?papers) ?area
WHERE {
  ?author sdm:submit ?paper .
  ?paper sdm:submitted_to ?venue ;
         sdm:has_publication ?publication .
  ?venue rdf:type ?venue_type ;
         sdm:has_area ?blank_ .
  ?venue_type rdfs:subClassOf sdm:Conference .
  ?blank_ rdf:type ?area .

  filter( contains( lcase(str(?area)) , "database"@en ) )
}
GROUP BY ?author ?area
```

Fig. 5: SPARQL query for finding all Authors with their publications for Database Conference

In order to get all authors, first we used "*sdm:submit*" *rdf:Property* to get all authors and their submitted papers. Then, we checked all those submitted papers; 1) who are submitted to some venue, 2) and papers which were published (i.e: they have "*sdm:has_publication*" *rdf:Property*). Since, we added only those papers as publications which were accepted by both reviewers, when we linked our ABOX and TBOX via Jena API, we will get all the accepted and published papers with "*sdm:has_publication*" *rdf:Property*.

After getting all the publications for a given author, we now need to see only publications which were submitted in the database conference. For this, we needed to first see papers who were published in conference proceedings. And according to our TBOX, all the *conference proceedings* will have *rdf:Property* "*sdm:has_publication*" and *rdfs:Class "sdm:Conference*". In order to get all conference proceedings, we checked all the venues which had a superclass of "*sdm:Conference*" (since, in our TBOX, we have four subclasses for "*sdm:Conference*" *rdfs:Class*). Then, we looked for papers which were published in a database conference. Here, we used *rdf:Property "sdm:has_area"* to get all conferences with their respective areas.

Now, we filtered areas which contained the word "database". We used contains and lcase to first convert all areas into lowercase string and then check if they contained the "database" keyword. As mentioned in this post, we used str to convert area IRI into string for checking.

Since, one author can have multiple publications for a database conference, so when displaying the output for this query, we grouped by results by authors and areas, and used group_concat (defined in SPARQL 1.1) along with a *distinct* keyword to avoid duplicated results, as mentioned here.