Student Online Teaching Advice Notice

The materials and content presented within this session are intended solely for use in a context of teaching and learning at Trinity.

Any session recorded for subsequent review is made available solely for the purpose of enhancing student learning.

Students should not edit or modify the recording in any way, nor disseminate it for use outside of a context of teaching and learning at Trinity.

Please be mindful of your physical environment and conscious of what may be captured by the device camera and microphone during videoconferencing calls.

Recorded materials will be handled in compliance with Trinity's statutory duties under the Universities Act, 1997 and in accordance with the University's policies and procedures.

Further information on data protection and best practice when using videoconferencing software is available at https://www.tcd.ie/info_compliance/data-protection/.

SPARQL

Part 2

SPARQL Protocol And RDF Query Language

Adapted from course notes of Dr. Christophe Debruyne & Dr. Rob Brennan

The WHERE CLAUSE

MORE ON SPARQL QUERY PATTERNS

Things you should already know

- **Multiple triple patterns** form conjunctions (ANDs), which are used:
- To retrieve multiple properties about a resource, eg

```
?fault ff:priority ?faultPriority .
?fault ff:timeStamp ?faultTimeStamp .
```

- To retrieve information on multiple resources, eg

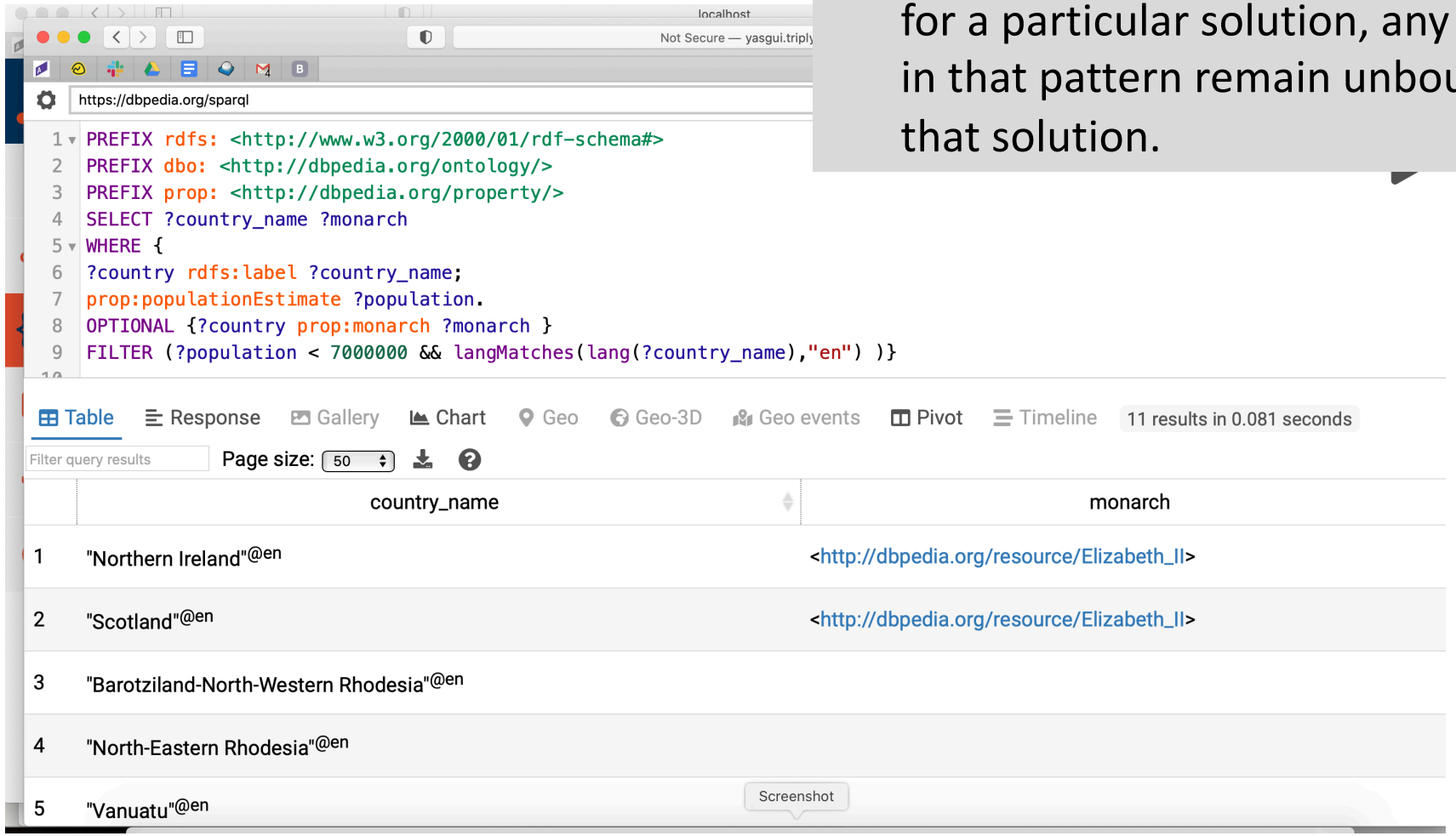
```
?fault ff:priority ?faultPriority .
?networkElement ff:manufacturer ?neManufacturer .
```

- By using a variable as an object of one triple and the subject of another, we traverse multiple links in the graph, eg

```
?fault ff:fromNetworkElement ?networkElement .
?networkElement ff:manufacturer ?neManufacturer .
```

Optional clause

- **`OPTIONAL`** tries to match a graph pattern, but does not fail the whole query if the optional match fails.
- If an **`OPTIONAL`** pattern fails to match for a particular solution, any variables in that pattern remain unbound for that solution.



Variable Assignment: BIND

Use of BIND ends the preceding basic graph pattern. The variable introduced by the BIND clause must not have been used in the graph pattern up to the point of use in BIND.

Yasgui - Triply

Query

https://dbpedia.org/sparql

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX db: <http://dbpedia.org/>
3  PREFIX dbp: <http://dbpedia.org/property/>
4  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5  PREFIX yago: <http://dbpedia.org/class/yago/>
6  PREFIX prop: <http://dbpedia.org/property/>
7  PREFIX dbpedia: <http://dbpedia.org/resource/>
8  SELECT ?statement
9  WHERE {?country rdfs:label  ?country_name;
10 prop:populationEstimate ?population.
11 BIND (CONCAT("The ",?country_name, " has population of ", ?population) AS ?statement).
12  FILTER (lang(?country_name) = "en")
13 }
```

Table | Response | Gallery | Chart | Geo | Geo-3D | Geo events | Pivot | Timeline | 46 results in 0.463 seconds

Filter query results | Page size: 50

	statement
1	The Caribbean Community has population of 16743693
2	The East African Federation has population of 153301178
3	The Central European Free Trade Agreement has population of 21907354

Screenshot

Subqueries

```
# PREFIXES OMITTED
SELECT DISTINCT ?country ?langfamily
WHERE{  ?lang dbo:languageFamily ?langfamily .
{SELECT DISTINCT ?country WHERE
{  ?country rdfs:label ?country_name;
          prop:populationEstimate ?population.
   ?country dbo:language ?lang .}
}} LIMIT 10
```

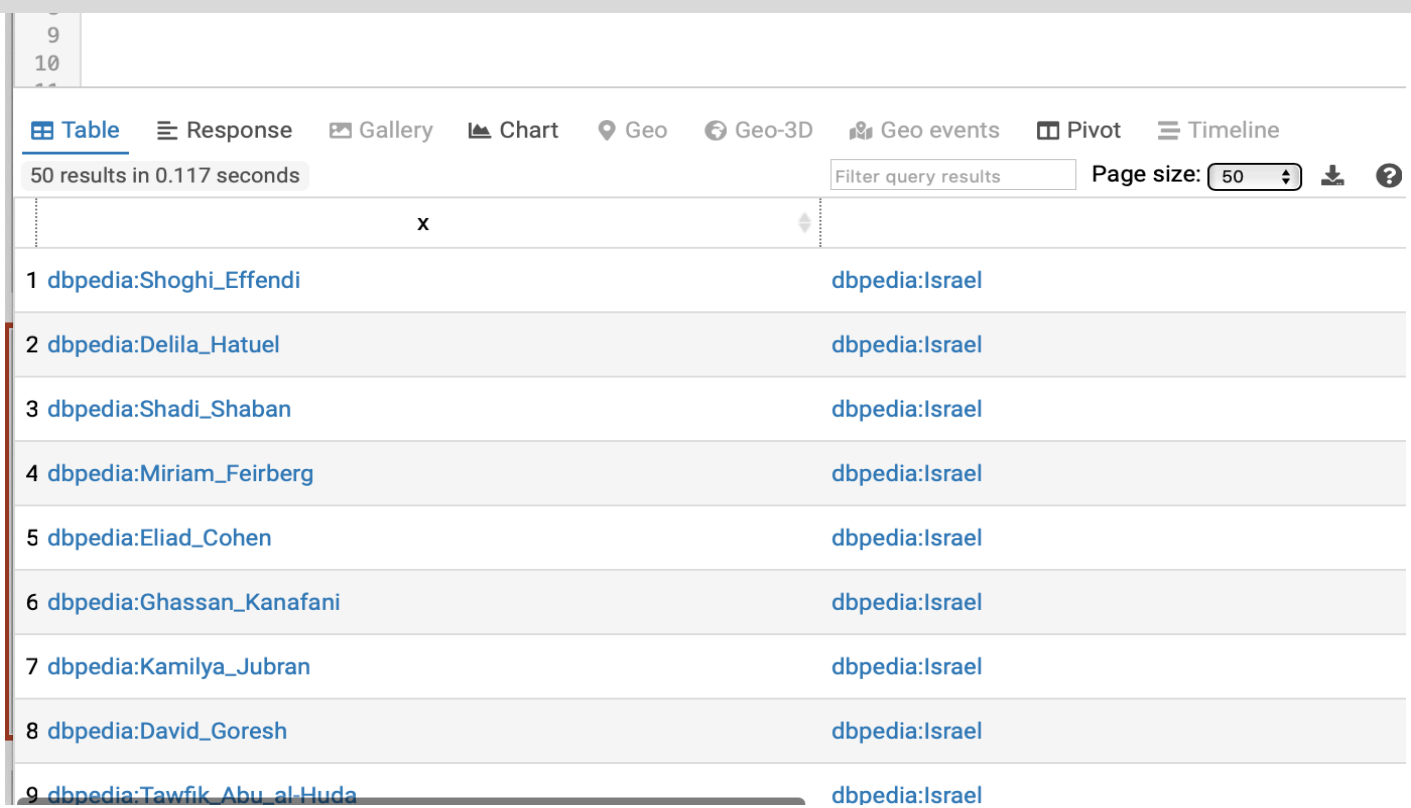
country	family
http://dbpedia.org/resource/Gulf_Cooperation_Council	http://dbpedia.org/resource/Algonquian_languages
http://dbpedia.org/resource/Gulf_Cooperation_Council	http://dbpedia.org/resource/Austronesian_languages
http://dbpedia.org/resource/Gulf_Cooperation_Council	http://dbpedia.org/resource/Bété_languages
http://dbpedia.org/resource/Gulf_Cooperation_Council	http://dbpedia.org/resource/Cree_language
http://dbpedia.org/resource/Gulf_Cooperation_Council	http://dbpedia.org/resource/Creole_language
http://dbpedia.org/resource/Gulf_Cooperation_Council	http://dbpedia.org/resource/Gur_languages
http://dbpedia.org/resource/Gulf_Cooperation_Council	http://dbpedia.org/resource/Hakka_Chinese
http://dbpedia.org/resource/Gulf_Cooperation_Council	http://dbpedia.org/resource/Kru_languages
http://dbpedia.org/resource/Gulf_Cooperation_Council	http://dbpedia.org/resource/Senufo_languages
http://dbpedia.org/resource/Gulf_Cooperation_Council	http://dbpedia.org/resource/Southeast_Asia
http://dbpedia.org/resource/Gulf_Cooperation_Council	http://dbpedia.org/resource/Tagalog_language
http://dbpedia.org/resource/Gulf_Cooperation_Council	http://dbpedia.org/resource/Turkish_language

Property Paths

Property paths used for querying arbitrary-length paths through the dataset graphs.

```
Find the country of any person's birthPlace

SELECT DISTINCT ?x  ?y
{?x dbo:birthPlace/dbo:country ?y .} LIMIT 50
```



Property Paths

Find the leader of the country of a person's birthplace
(or three links away).

```
SELECT DISTINCT ?x  ?y
{?x dbo:birthPlace/dbo:country/dbo:leader ?y .}
LIMIT 50
```

This is equivalent to…

```
SELECT ?x ?y {
 ?x dbo:birthPlace ?z1.
 ?z1 dbo:country ?z2 .
 ?z2 dbo:leader ?y .
} LIMIT 50
```

Property Paths

Use ***, +, ?** (ala DTDs) to find connections in arbitrary length paths.

Arbitrary Length

`Example` finding all the the possible types of a resource, including supertypes of resource

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

SELECT ?x ?type { ?x rdf:type/rdfs:subClassOf* ?type }
```

<#verb-group>	<http://www.w3.org/2002/07/owl#SymmetricProperty>
<#verb-group>	<http://www.w3.org/2002/07/owl#ObjectProperty>
<#verb-group>	rdf:Property
<http://dbpedia.org/ontology/deathDate>	<http://www.w3.org/2002/07/owl#FunctionalProperty>
<http://dbpedia.org/ontology/deathDate>	rdf:Property
<http://dbpedia.org/ontology/deathDate>	<http://www.w3.org/2002/07/owl#FunctionalProperty>
<http://dbpedia.org/ontology/deathDate>	rdf:Property
<http://dbpedia.org/ontology/birthDate>	<http://www.w3.org/2002/07/owl#FunctionalProperty>
<http://dbpedia.org/ontology/birthDate>	rdf:Property
<http://dbpedia.org/ontology/birthDate>	<http://www.w3.org/2002/07/owl#FunctionalProperty>

Property Paths: one or other on path

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT *
{ ?book dc:title|rdfs:label ?displayString }
```

369	<#WN30-102715229>	antenna
370	<#WN30-104843270>	antenna
371	<#WN30-102584915>	antenna
372	<http://www.openlinksw.com/virtpivot/icons/SurveyCollection>	SurveyCollection (DisplayClass)
373	<http://www.openlinksw.com/virtpivot/icons/SurveyCollectionIcon>	SurveyCollection icon
374	<http://www.openlinksw.com/virtpivot/icons/Weblog>	Weblog (DisplayClass)
375	<http://www.openlinksw.com/virtpivot/icons/WeblogIcon>	Weblog icon
376	<http://www.openlinksw.com/virtpivot/icons/Wiki>	Wiki (DisplayClass)
377	<http://www.openlinksw.com/virtpivot/icons/WikiIcon>	Wiki icon
378	<http://www.openlinksw.com/virtpivot/icons/ElectronicGood>	ElectronicGood (DisplayClass)
379	<http://www.openlinksw.com/virtpivot/icons/ElectronicGoodIcon>	ElectronicGood icon
380	<http://www.openlinksw.com/virtpivot/icons/Camera>	Camera (DisplayClass)
381	<http://www.openlinksw.com/virtpivot/icons/CameraIcon>	Camera icon
382	<http://www.openlinksw.com/virtpivot/icons/Computer>	Computer (DisplayClass)
383	<http://www.openlinksw.com/virtpivot/icons/ComputerIcon>	Computer icon
384	<http://www.openlinksw.com/virtpivot/icons/LaptopComputer>	LaptopComputer (DisplayClass)
385	<http://www.openlinksw.com/virtpivot/icons/LaptopComputerIcon>	LaptopComputer icon
386	<http://www.openlinksw.com/virtpivot/icons/Monitor>	Monitor (DisplayClass)

TryItOut: Property Path

```
PREFIX db: <http://dbpedia.org/>
PREFIX dbpr: <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?person ?name WHERE {
  ?person dbo:birthPlace <http://dbpedia.org/resource/Dublin> .
  ?person dbp:name ?name.
}
```

- Add a line to the query above to use a predicate path to get the **dbp:fullName** of **dbo:almamater** (?college) that a person went to where it exists, otherwise just return the resource and name of person

- Hint:

  Recall OPTIONAL CLAUSE from earlier

TryItOut: Solution and partial results

```
PREFIX db: <http://dbpedia.org/>
PREFIX dbpr: <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT ?person ?name ?college WHERE {
  ?person dbo:birthPlace <http://dbpedia.org/resource/Dublin> .
  ?person dbp:name ?name.
  OPTIONAL {?person dbo:almaMater/dbp:fullName ?college.}
}
```

	person	name	college
1	db:resource/Oscar_Wilde	"Oscar Wilde"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>	"The Provost, Fellows and Scholars of the College of the Holy and Undivided Trinity of Queen Elizabeth near Dublin"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>
2	db:resource/Jonathan_Swift	"Jonathan Swift"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>	"The Provost, Fellows and Scholars of the College of the Holy and Undivided Trinity of Queen Elizabeth near Dublin"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>
3	db:resource/Brendan_Kelly_(actor)	"Brendan Kelly"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>	
4	db:resource/Freeman_Wills_Crofts	"Freeman Wills Crofts"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>	
5	db:resource/Tom_Dunne	"Tom Dunne"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>	

Screenshot

Union

- **UNION** is useful by combining two or more *graph patterns*.
- It is not restricted to triple patterns.

```
SELECT ?country_name ?population
WHERE {
  {?country rdfs:label ?country_name;
   prop:populationEstimate ?population;
   prop:monarch ?monarch .
FILTER (?population > 50000000
&&langMatches(lang(?country_name),"en") )
}
  UNION
   {?country rdfs:label ?country_name;
   prop:populationEstimate ?population;
   prop:monarch ?monarch .
FILTER (?population < 2000000
&&langMatches(lang(?country_name),"en") )
}}
```

	country_name	population
1	"England"@en	"54786300"^^xsd:integer
2	"Northern Ireland"@en	"1864000"^^xsd:integer

TryItOut: Union

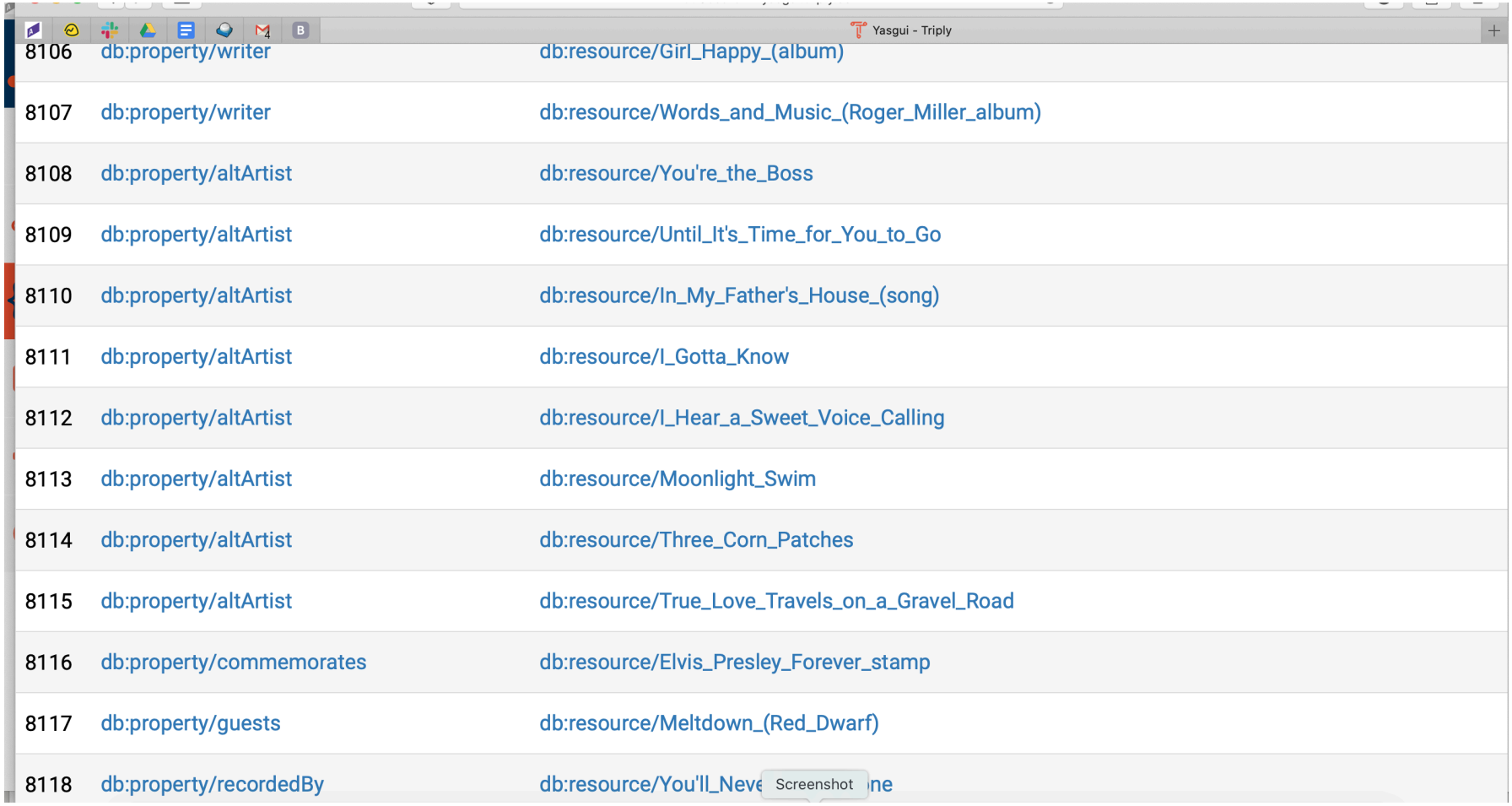
Create a query to retrieve all properties about Elvis using UNION

In other words: all predicates and objects where the Elvis resource is the subject, and all predicates and subjects where the Elvis resource is the object.

Use <http://dbpedia.org/resource/Elvis_Presley>

TryItOut: Union

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX db: <http://dbpedia.org/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX yago: <http://dbpedia.org/class/yago/>
PREFIX prop: <http://dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/resource/>
SELECT DISTINCT ?prop ?val
{
{ <http://dbpedia.org/resource/Elvis_Presley> ?prop ?val. }
UNION
{ ?val ?prop <http://dbpedia.org/resource/Elvis_Presley>. }
}
```


Yasgui - Triply
8106 db:property/writer db:resource/Girl_Happy_(album)
8107 db:property/writer db:resource/Words_and_Music_(Roger_Miller_album)
8108 db:property/altArtist db:resource/You're_the_Boss
8109 db:property/altArtist db:resource/Until_It's_Time_for_You_to_Go
8110 db:property/altArtist db:resource/In_My_Father's_House_(song)
8111 db:property/altArtist db:resource/I_Gotta_Know
8112 db:property/altArtist db:resource/I_Hear_a_Sweet_Voice_Calling
8113 db:property/altArtist db:resource/Moonlight_Swim
8114 db:property/altArtist db:resource/Three_Corn_Patches
8115 db:property/altArtist db:resource/True_Love_Travels_on_a_Gravel_Road
8116 db:property/commemorates db:resource/Elvis_Presley_Forever_stamp
8117 db:property/guests db:resource/Meltdown_(Red_Dwarf)
8118 db:property/recordedBy db:resource/You'll_Neve
Screenshot
ne

Union

The problem, however, is that one ‘loses’ the direction of the properties.

Create a query that adds “is … of” to a property of which Elvis_Presley was the object.

For example

“IS http://dbpedia.org/ontology/wikiPageWikiLink OF”

Tips:

Consider using a clause that binds to a new variable and a function that combines strings together

Use resource <http://dbpedia.org/resource/Elvis_Presley>

Union

```
SELECT DISTINCT ?prop ?v
{ ?v ?p <http://dbpedia.org/resource/Elvis_Presley>.
BIND(CONCAT("IS ", ?p, " OF") AS ?prop).  }
```

prop	v
IS http://dbpedia.org/ontology/wikiPageWikiLink OF	http://dbpedia.org/resource/1935
IS http://dbpedia.org/ontology/wikiPageWikiLink OF	http://dbpedia.org/resource/1950s
IS http://dbpedia.org/ontology/wikiPageWikiLink OF	http://dbpedia.org/resource/1954
IS http://dbpedia.org/ontology/wikiPageWikiLink OF	http://dbpedia.org/resource/1955

Aggregates

- Aggregate queries process query results by dividing the solutions into groups, and then performing summary calculations on those groups.
- As in SQL, the GROUP BY clause specifies the key variable(s) to use to partition the solutions into groups.
- SPARQL 1.1 defines aggregate functions too: COUNT, MIN, MAX, SUM, AVG, GROUP_CONCAT, SAMPLE

Example Aggregate

```
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX db: <http://dbpedia.org/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT ?language (SUM(?population) AS ?totalPopulation)
WHERE {
?country dbp:populationEstimate ?population .
?country dbp:languages ?language.
}
GROUP BY ?language
```

We can introduce a **HAVING** clause to GROUP BY line to filter the results of the query **after** applying aggregates.

For example
HAVING (SUM(?population)>6000000)

Table | Response | Gallery | Chart | Geo | Geo-3D | Geo events | Pivot | Timeline | 10 results in 0.256 seconds | Filter query results | Page size: 50

	language	totalPopulation
1	"English"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>	"471964016"^^<http://www.w3.org/2001/XMLSchema#integer>
2	db:resource/Scots_language	"5373000"^^<http://www.w3.org/2001/XMLSchema#integer>
3	db:resource/Scottish_Gaelic	"5373000"^^<http://www.w3.org/2001/XMLSchema#integer>
4	db:resource/British_Sign_Language	"5373000"^^<http://www.w3.org/2001/XMLSchema#integer>
5	db:resource/Ulster_Scots_dialects	"1864000"^^<http://www.w3.org/2001/XMLSchema#integer>
6	db:resource/English_language	"62023300"^^<http://www.w3.org/2001/XMLSchema#integer>
7	db:resource/Cornish_language	"54786300"^^<http://www.w3.org/2001/XMLSchema#integer>
8	"French"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>	"471964016"^^<http://www.w3.org/2001/XMLSchema#integer>
9	db:resource/Irish_language	"1864000"^^<http://www.w3.org/2001/XMLSchema#integer>
10	"Spanish"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>	"471964016"^^<http://www.w3.org/2001/XMLSchema#integer>

Showing 1 to 10 of 10 entries < 1 >

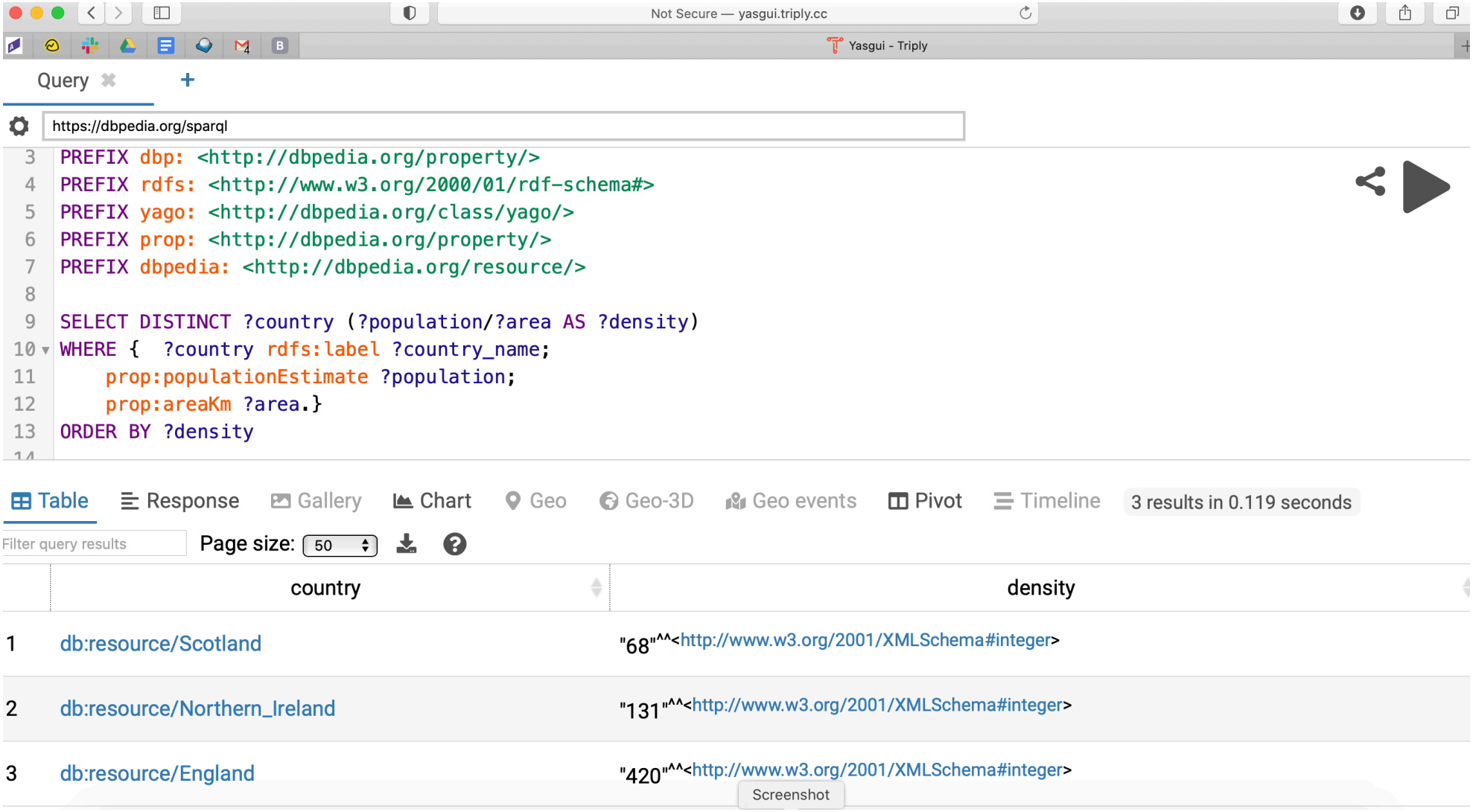
Screenshot

MORE ON THE RESULT CLAUSE

Projected Expressions

- SPARQL 1.1 Query adds the ability for query results to contain values derived from constants, function calls, or other expressions in the SELECT list.
- Allows for arbitrary expressions to be used for columns in a query's result set.
- Projected expressions must be in parentheses and must be given an alias using the AS keyword.
- *Note:* [] in a query acts as an unnamed variable.

Projected Expressions



SPARQL Result Clause

- SPARQL has 4 result forms:
  - SELECT – Return a table of results.
  - CONSTRUCT – Return an RDF graph, based on a template in the query.
  - DESCRIBE – Return an RDF graph, based on what the query processor is configured to return.
  - ASK – Ask a boolean query.
- The SELECT form directly returns a table
- DESCRIBE and CONSTRUCT use the outcome of matching to build RDF graphs

Slide source: Christophe Debruyne

SPARQL CONSTRUCT

- Construct graphs from graphs based on a query
- Useful for – one-step – transformation between vocabularies

```
CONSTRUCT {?s <http://example.com/covid#FrontLineWorkers> ?o.}
WHERE {?s  <http://ontologies.geohive.ie/covid#healthcareWorkerCases> ?o .}
```

CONSTRUCT template.

	subject	predicate	object
1	http://data.geohive.ie/resource/covid/statpro-file/2020-03-16	http://example.com/covid#FrontLineWorkers	"59"^^xsd:integer
2	http://data.geohive.ie/resource/covid/statpro-file/2020-03-17	http://example.com/covid#FrontLineWorkers	"84"^^xsd:integer
3	http://data.geohive.ie/resource/covid/statpro-file/2020-03-18	http://example.com/covid#FrontLineWorkers	"114"^^xsd:integer
4	http://data.geohive.ie/resource/covid/statpro-file/2020-03-19	http://example.com/covid#FrontLineWorkers	"147"^^xsd:integer
5	http://data.geohive.ie/resource/covid/statpro-file/2020-03-20	http://example.com/covid#FrontLineWorkers	"159"^^xsd:integer
6	http://data.geohive.ie/resource/covid/statpro-file/2020-03-21	http://example.com/covid#FrontLineWorkers	"208"^^xsd:integer
7	http://data.geohive.ie/resource/covid/statpro-file/2020-03-22	http://example.com/covid#FrontLineWorkers	"247"^^xsd:integer

keyboard shortcuts

SPARQL DESCRIBE

- DESCRIBE form takes each of the resources identified in a solution, together with any resources directly named by IRI, and assembles a single RDF graph by taking a "description" which can come from any information available including the target RDF Dataset.
- The description is determined by the query service.

Query +

https://dbpedia.org/sparql

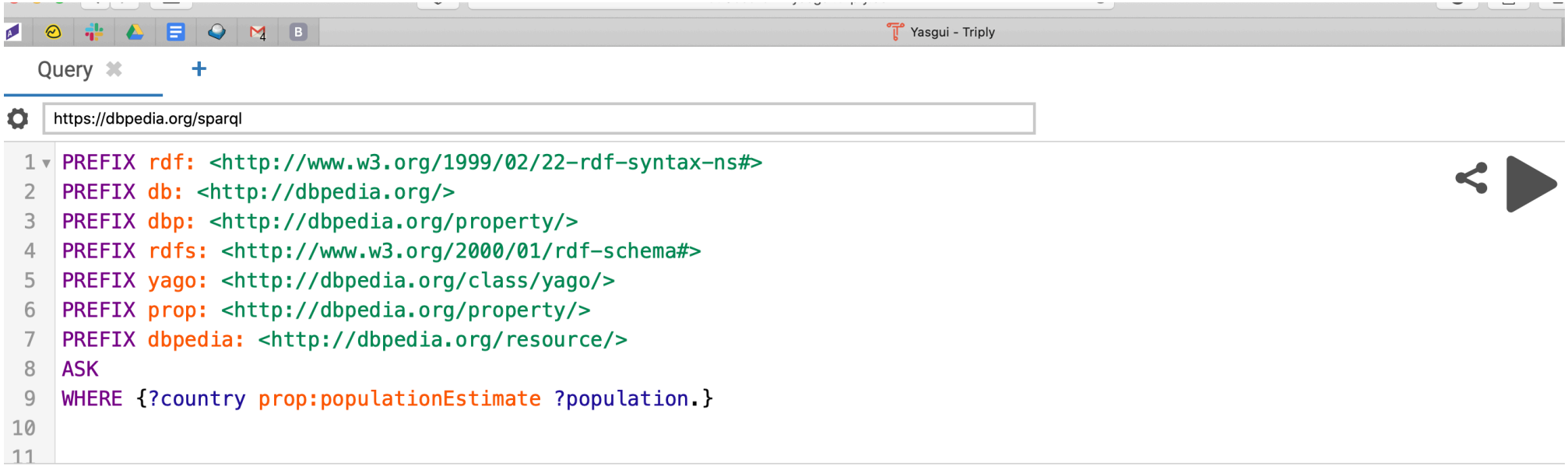
```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX db: <http://dbpedia.org/>
3  PREFIX dbp: <http://dbpedia.org/property/>
4  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5  PREFIX yago: <http://dbpedia.org/class/yago/>
6  PREFIX prop: <http://dbpedia.org/property/>
7  PREFIX dbpedia: <http://dbpedia.org/resource/>
8
9  DESCRIBE <http://dbpedia.org/resource/Elvis>
10
11
```

Yasgui - Triply

28	db:resource/The_Cassettes	db:ontology/wikiPageWikiLink	db:resource/Elvis	<>
29	db:resource/The_Crackerjacks_(band)	db:ontology/wikiPageWikiLink	db:resource/Elvis	<>
30	db:resource/WCVG	db:ontology/wikiPageWikiLink	db:resource/Elvis	<>
31	db:resource/TimeSplitters_2	db:ontology/wikiPageWikiLink	db:resource/Elvis	<>
32	db:resource/List_of_Archie_Comics_publications	db:ontology/wikiPageWikiLink	db:resource/Elvis	<>
33	db:resource/Elvis	rdfs:label	"Elvis"@en	<>
34	db:resource/Elvis	db:ontology/wikiPageID	"1796642"^^<http://www.w3.org/2001/XMLSchema#integer>	<>
35	db:resource/Elvis	db:ontology/wikiPageRevisionID	"734551737"^^<http://www.w3.org/2001/XMLSchema#integer>	<>
36	db:resource/Elvis	db:ontology/wikiPageWikiLink	db:resource/Elvis_Presley	<>

SPARQL ASK

- ASK is used to test whether or not a query pattern has a solution
- Returns True or False



FROM CLAUSE

DATASETS/NAMED GRAPHS

RDF Datasets/Named Graphs

- RDF Dataset = unit queried by a SPARQL query
  - The default graph + named graphs
- Graph matching works on 1 graph
  - By default this is the default graph but you can specify (via the `GRAPH` keyword) a specific graph or graphs
- RDF datasets are flexible, there are no limitations - one graph can be included twice under different names, or some graphs may share triples with others.
  - default graph the union of all named graphs

=> Named graphs lead to the concept of Quads:

<graph name>, <subject>, <predicate>, <object>

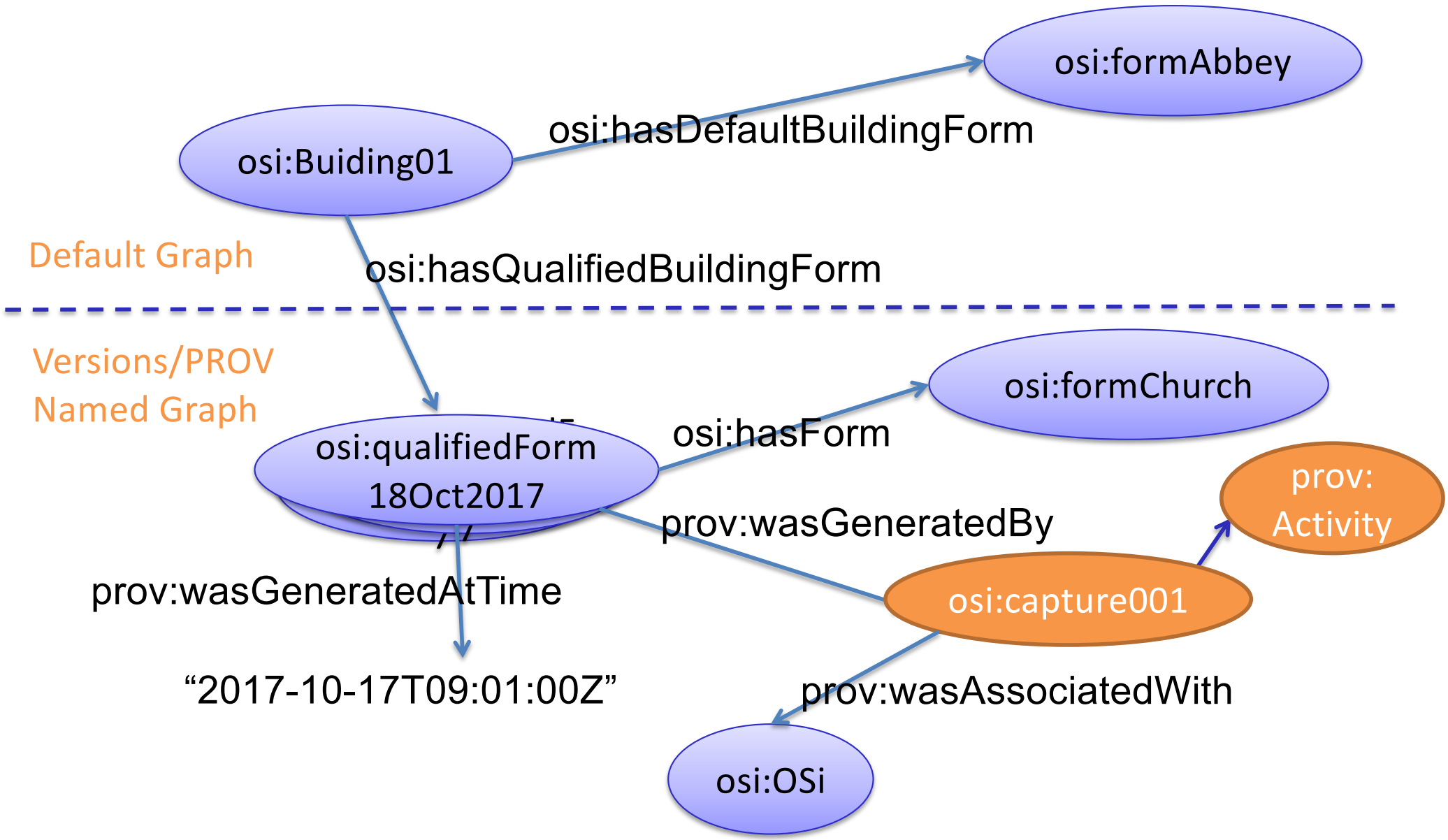
Slide source: Rob Brennan

Why use Named Graphs?

- To identify a subset of RDF triples in a triple-store
- To split the query space
- To store related data in one logical location
- To identify the source of origin for a set of triples
- To separate metadata from data
- To separate the schema from the instances
- For versioning support
- For access control
- To make management of a large dataset easier

Slide source: Rob Brennan

Example Use of Named Graph

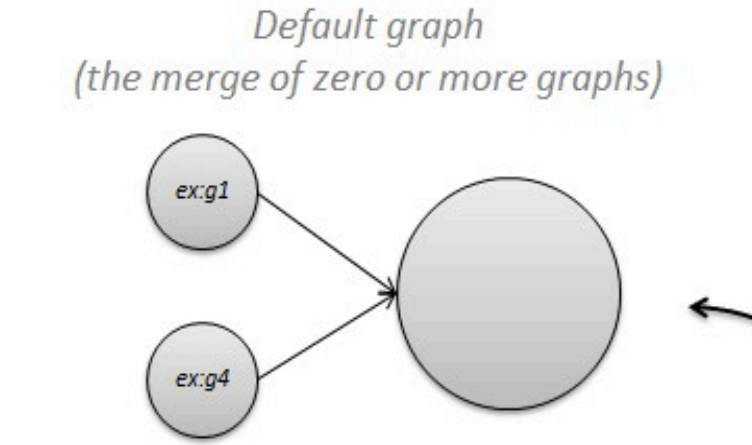


Named Graphs

- Instead of triples, we have quads
- Graph can serve as ‘context’
- Queries may specify the datasets to be used for matching
  - **`FROM`** clauses to refer to default graphs
  - **`FROM NAMED`** clauses to refer to named graphs
  - If **`FROM NAMED`** clauses are provided without a **`FROM`** clause, the default empty graph is assumed to be used.

Slide source: Rob Brennan

Named Graphs




```
PREFIX ex: <…>
SELECT …
FROM ex:g1
FROM ex:g4
FROM NAMED ex:g1
FROM NAMED ex:g2
FROM NAMED ex:g3
WHERE {
  … A …
  GRAPH ex:g3 {
    … B …
  }
  GRAPH ?graph {
    … C …
  }
}
```

Adapted from “SPARQL by Example” by Feigenbaum and Prud’hommeaux

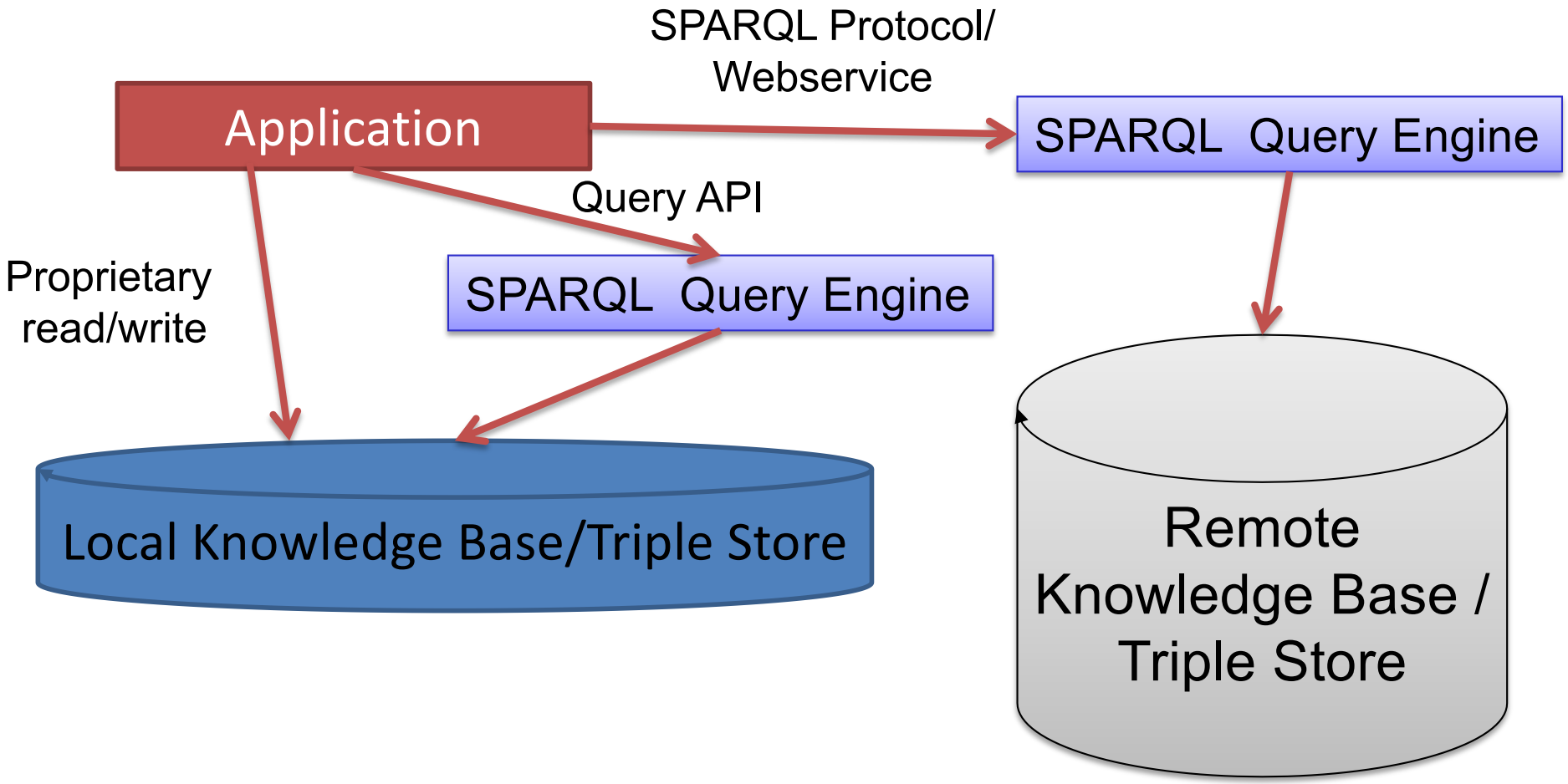
Example

```
SELECT ?src ?x
FROM NAMED <http://example.com/covid/county>
FROM NAMED <http://example.com/covid/county_v2>
WHERE
 {
  GRAPH ?src
  { ?x  <http://ontologies.geohive.ie/covid#confirmedCovidCases> ?s .
   ?x  <http://ontologies.geohive.ie/covid#confirmedCovidCases> ?s .
  }
 }
```

1	http://example.com/covid/county	http://data.geo
2	http://example.com/covid/county_v2	http://data.geo
3	http://example.com/covid/county	http://data.geohive.ie/resource/covid/countystat/2020-05-01Limerick
4	http://example.com/covid/county_v2	http://data.geohive.ie/resource/covid/countystat/2020-05-01Limerick
5	http://example.com/covid/county	http://data.geohive.ie/resource/covid/countystat/2020-05-23Cork
6	http://example.com/covid/county_v2	http://data.geohive.ie/resource/covid/countystat/2020-05-23Cork
7	http://example.com/covid/county	http://data.geohive.ie/resource/covid/countystat/2020-07-19Kildare
8	http://example.com/covid/county_v2	http://data.geohive.ie/resource/covid/countystat/2020-07-19Kildare
9	http://example.com/covid/county	http://data.geohive.ie/resource/covid/countystat/2020-05-04Offaly
10	http://example.com/covid/county_v2	http://data.geohive.ie/resource/covid/countystat/2020-05-04Offaly
11	http://example.com/covid/county	http://data.geohive.ie/resource/covid/countystat/2020-06-22Westmeath
12	http://example.com/covid/county_v2	http://data.geohive.ie/resource/covid/countystat/2020-06-22Westmeath

Screenshot

Recall: Where SPARQL fits



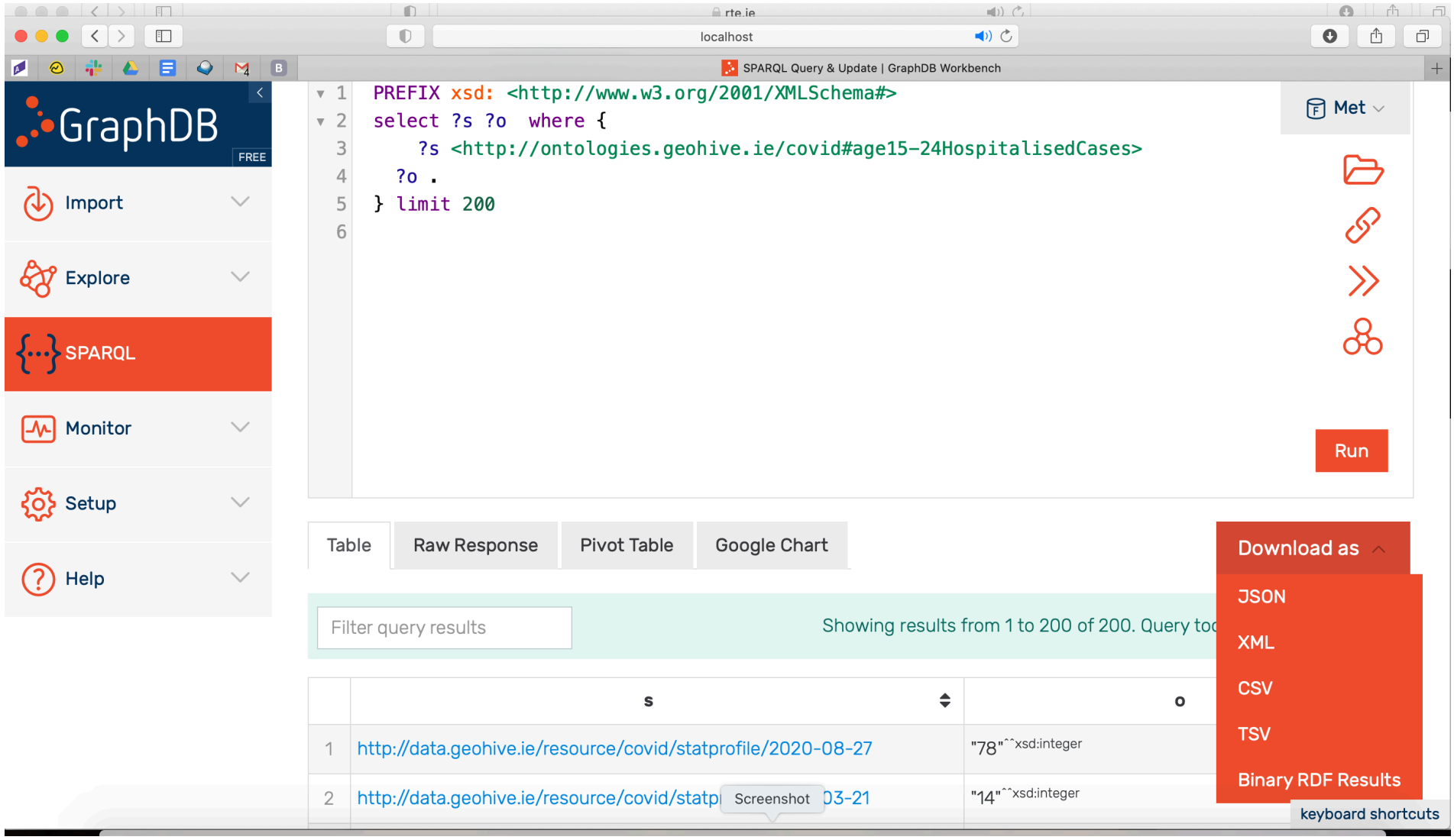
Slide source: Christophe Debruyne

Recall: SPARQL Architecture and Endpoints

- SPARQL queries are executed against **RDF datasets**
  - Consisting of one or more RDF graphs
- A **SPARQL endpoint** accepts queries and returns results via HTTP
- The results returned/rendered in a variety of formats:
  - SPARQL specifies an **XML** vocabulary for result sets
  - A JSON "port" of the XML vocabulary
  - Certain SPARQL result clauses trigger **RDF** responses
  - **HTML** often as an XSLT transformation of the XML
  - **CSV, TSV etc.**

Slide source: Rob Brennan

https://graphdb.ontotext.com



Returning SPARQL Results as XML

- Schema-defined format acts as a bridge between RDF queries and XML tools and libraries.
- There are a number of potential uses for this capability. - - Could transform the results of a SPARQL query into a Web page or RSS feed via XSLT,
- Access the results via XPath, or
- Return the result document to an AJAX client.
- Typically this is a command line switch for the SPARQL engine

Example XML Query Result

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
<head>
  <variable name="x"/>
  <variable name="hpage"/>
  <variable name="name"/>
</head>
<results>
  <result>
      <binding name="x">
        <bnode>r2</bnode>
      </binding>
      <binding name="hpage">
        <uri>http://work.example.org/bob/</uri>
      </binding>
      <binding name="name">
        <literal xml:lang="en">Bob</literal>
      </binding>
  </result>
... </results>
</sparql>
```

<- Query result document element

<- Query result sub-element 1, header

<- sequence of elements describing the set of Query Variable names in the Solution Sequence

<- Query result sub-element 2, results (always present, even if empty)

<- Query solution child-element, result

Blank node binding

RDF URI binding

RDF literal binding

<- child elements for each *Query Variable* that appears in the solution. It is used to record how the query variables bind to RDF Terms.

Federated Queries over 2 endpoints (theory)

The SERVICE keyword is used to send part of a query against a remote SPARQL endpoint.
Find the birth dates of all of the actors in *Star Trek: The Motion Picture*

```
PREFIX movie: <http://data.linkedmdb.org/resource/movie/>
PREFIX dbpedia: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?actor_name ?birth_date FROM
<http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf> # placeholder
graph
WHERE {
SERVICE <http://data.linkedmdb.org/sparql> {
    <http://data.linkedmdb.org/resource/film/675> movie:actor
?actor .
    ?actor movie:actor_name ?actor_name }
SERVICE <http://dbpedia.org/sparql> {
    ?actor2 a dbpedia:Actor ;
    foaf:name ?actor_name_en ;
    dbpedia:birthDate ?birth_date .    FILTER(STR(?actor_name_en) =
?actor_name) } }
```

- (The FILTER is necessary because names in dbpedia have language tags, while names in LinkedMDB do not.)

Federated Queries (public endpoints)

Difficult to find reliable public endpoints that allow federated queries, and it's even more difficult to find endpoints with "nice enough" timeout periods

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT * WHERE {
[] a ?type .
SERVICE <https://environment.data.gov.uk/sparql/bwq/query>
{[] a ?type .}} LIMIT 2
```

Using http://data.ordnancesurvey.co.uk/datasets/os-linked-data/apis/sparql as endpoint in YASGUI

Finds two types that are common in both graphs

SPARQL 1.1 Update features

- SPARQL 1.1 Update is a language for managing and updating RDF graphs.

- `INSERT DATA { triples }`
- `DELETE DATA { triples }`
- `[ DELETE { template } ] [ INSERT { template } ] WHERE { pattern }`
- `LOAD uri [ INTO GRAPH uri ]`
- `CLEAR GRAPH uri`
- `CREATE GRAPH uri`
- `DROP GRAPH uri`

Slide source: Rob Brennan

Useful References

- Feigenbaum and Prud'hommeaux. SPARQL by Example.
  http://www.cambridgesemantics.com/semantic-university/sparql-by-example

- SPARQL 1.1 Overview
  https://www.w3.org/TR/sparql11-overview/

- SPARQL 1.1 Specification
  https://www.w3.org/TR/sparql11-query/

Negation in SPARQL 1.0

Find the people that do not contain a URL with the `rdfs:seeAlso` predicate:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?name ?url
WHERE {
  ?person a foaf:Person ; foaf:name ?name .
  OPTIONAL { ?person rdfs:seeAlso ?url }
  FILTER(!bound(?url))
}
```

Negation in SPARQL 1.1

NOT EXISTS and MINUS represent two ways of thinking about negation.

- The first based on testing whether a pattern exists in the data, given the bindings already determined by the query pattern.
- The second based on removing matches based on the evaluation of two patterns.

Negation in SPARQL 1.1

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?name ?url
WHERE {
  ?person a foaf:Person ; foaf:name ?name .
  FILTER(NOT EXISTS { ?person rdfs:seeAlso ?url })
}
```

SPARQL 1.1 NOT EXISTS filter uses the bindings from a **solution** to test whether a given pattern exists.

Negation in SPARQL 1.1

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?name ?url
WHERE {
  ?person a foaf:Person ; foaf:name ?name .
  MINUS { ?person rdfs:seeAlso ?url }
}
```

SPARQL 1.1 MINUS graph pattern clause is a binary operator that **removes bindings that match the right-hand side**.