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

SPARQL **P**rotocol **A**nd **R**DF **Q**uery **L**anguage

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

Overview

- RDF = data model for the web of data
  - Enables data to be decentralized and distributed.
  - RDF models can be merged together easily.
- SPARQL = searching and retrieval of RDF data (and more recently updates)
  - Not just keyword matching, can query properties/relationships between classes or instances
    - E.g. Find all the instances that have the “has Brother” relationship
  - Due to RDF model, can query schema as well as data
    - E.g find all the classes which are sub-classes of “Person”
  - SQL power for the whole (semantic) Web!
    - SELECT <something> returns tabular data
  - Perform complex joins of disparate databases in a single, simple query
  - Transform RDF data from one vocabulary to another
  - With SPARQL 1.1, **updates as well**

Slide source: Rob Brennan

SPARQL

- Stands for *SPARQL Protocol and RDF Query Language*
  - Recursive acronym
- SPARQL 1.1 is a W3C Recommendation since March 2013 (http://www.w3.org/TR/sparql11-overview/)
- SPARQL allows us to:
  - Pull values from structured and semi-structured data
  - Explore data by querying unknown relationships
  - Perform complex joins over different graphs in a single, simple query
  - Even over different endpoints in a single, simple query
  - Transform RDF data from one vocabulary to another
  - …

Slide source: Christophe Debruyne

Original Requirements

- Supports application query types:
  - Lexical
  - Simple semantics based
    - Retrieve instances of a class / types of an instance
  - “Arbitrary" conjunctive query
    - Find all ?x, ?y such that ?x loves ?y and ?y loves ?z

      (like SQL select-from-where queries in which the where clause conditions are constructed from column names and constants using no comparison operators other than "=", combined using "and“)
- Implementable
- Predictable results
- Modular semantics

Slide source: Christophe Debruyne

Background – W3C standard

SPARQL 1.0

- January 2008
- SPARQL 1.0 Query Language
- SPARQL 1.0 Protocol
- Results in XML
- Results in JSON

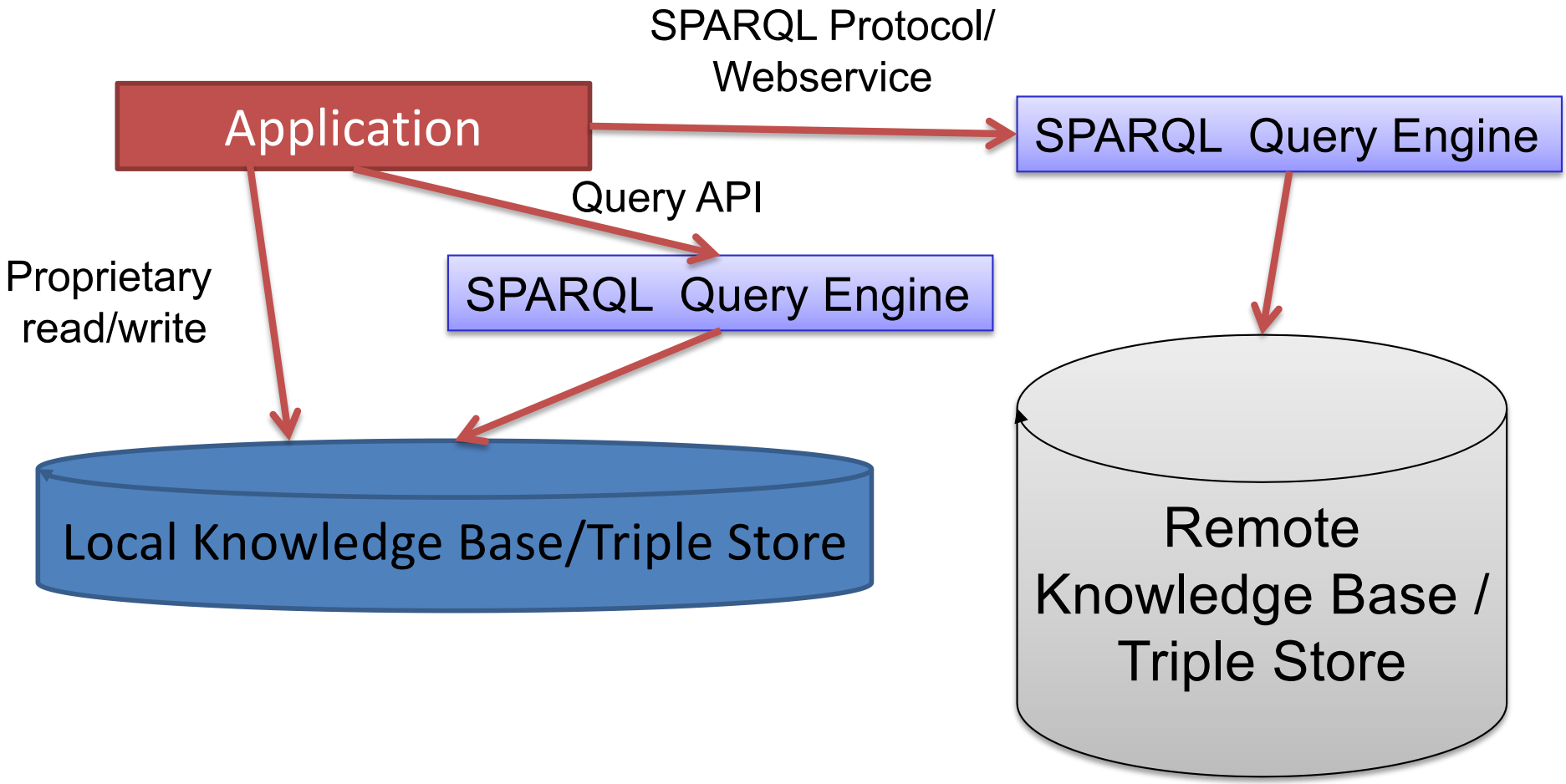
SPARQL 1.1

- March 2013
- Several updates
- Additional Features
  - Update (insert, del, modify)
  - Graph Store HTTP protocol
  - Service Descriptions
  - Entailments
  - Basic Federated Query
  - Results in CSV, TSV

Slide source: Christophe Debruyne

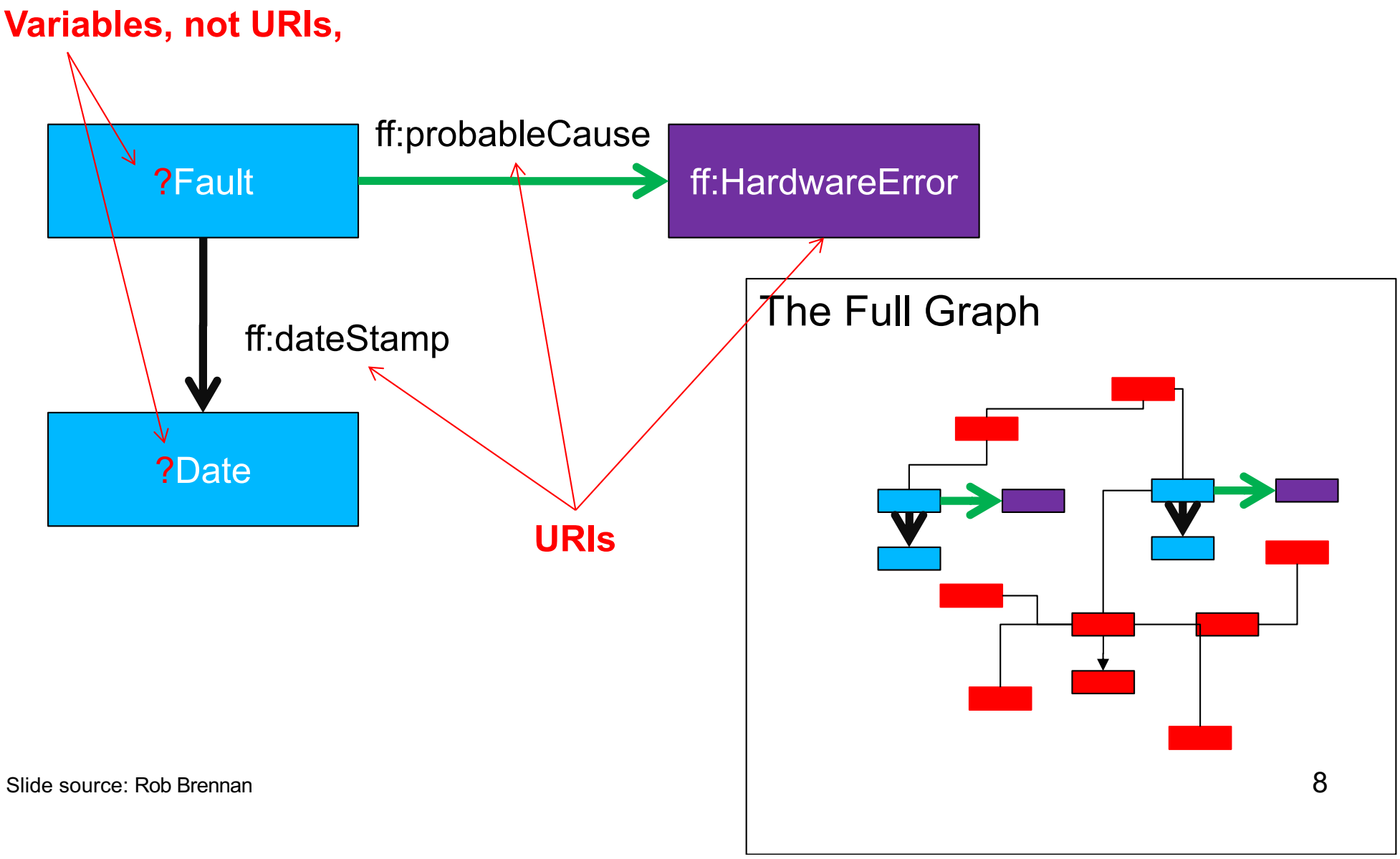
6

Where SPARQL fits



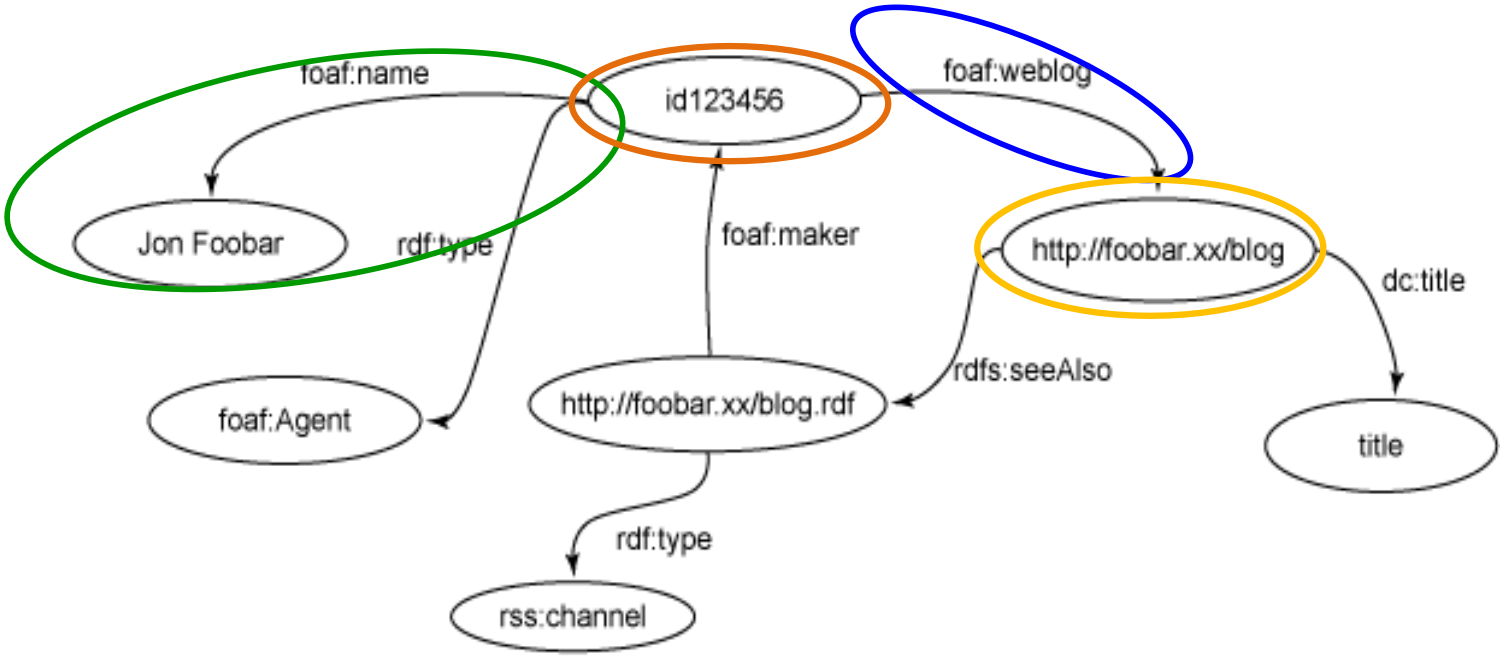
Slide source: Christophe Debruyne

SPARQL based on Graph Matching



Slide source: Rob Brennan

Query in action




```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?url
WHERE  {
?contributor      foaf:name     "Jon Foobar" .
?contributor      foaf:weblog   ?url .
}
```

<- NAMESPACE

<- Triple patterns using Turtle syntax

Explaining the Action in the Query

1. WHERE clause - the triples together comprise a graph pattern.

2. Query tries to match the graph pattern to any part of the graph in the triplestore (the model).

3. Each match that binds the graph pattern variables to the model's nodes becomes a “query solution”.

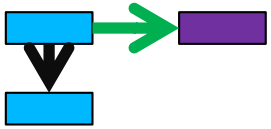
4. The values of the variables named in the SELECT clause become part of the query results.

- In the example, the first triple in the WHERE clause's graph pattern matches a node with a foaf:name property of "Jon Foobar," and binds it to the variable named contributor.
- In the bloggers.rdf model, contributor will match the foaf:Agent blank-node at the top of the figure (id123456).
- The graph pattern's second triple matches the object of the contributor's foaf:weblog property.
- This is bound to the ?url variable, forming a query solution.

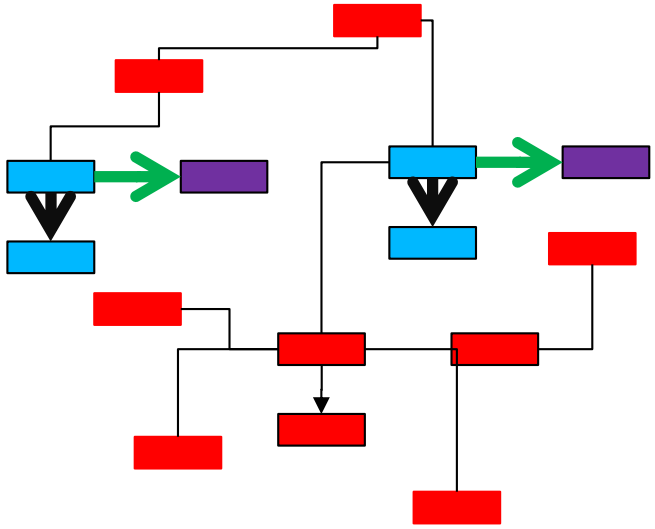
Query Results Pipeline

WHERE CLAUSE

1. Triple Pattern



2. Graph Matching



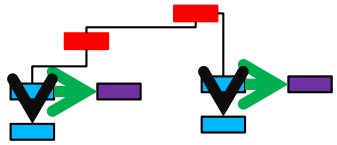
RESULT CLAUSE & API

3. Query Results

A. Table

Fault	Date
1001	29 May
1002	30 May

B. New Graph



C. XML

< < ../> .../>

D. JSON, CSV etc.

Structure of a SPARQL Query

A SPARQL query comprises, in order:

- **Prefix declarations** for URI shorthands

Namespace shortened for brevity

```
# prefix declarations
PREFIX foaf: <http://.../0.1/>
...
```

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

Structure of a SPARQL Query

A SPARQL query comprises, in order:

-
- The **query pattern**, specifying what to query for in the underlying dataset

```
# query pattern
WHERE {
    ...
}
```

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

Structure of a SPARQL Query

A SPARQL query comprises, in order:

-

- **Query modifiers**, slicing, ordering, and otherwise rearranging query results

```
# query modifiers
ORDER BY ...
```

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

Structure of a SPARQL Query

A SPARQL query comprises, in order:

-
- A **Result Clause**, identifying what information to return from the query

```
# result clause
SELECT ...
```

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

Structure of a SPARQL Query

A SPARQL query comprises, in order:

-

- **Dataset definition**, stating what RDF graph(s) are being queried

```
# dataset definition
FROM ...
```

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

Structure of a SPARQL Query

A SPARQL query comprises, in order:

- **Prefix declarations** for URI shorthands
- A **Result Clause**, identifying what information to return from the query
- **Dataset definition**, stating what RDF graph(s) are being queried
- The **query pattern**, specifying what to query for in the underlying dataset
- **Query modifiers**, slicing, ordering, and otherwise rearranging query results

Namespace shortened for brevity

```
# prefix declarations
PREFIX foaf: <http://.../0.1/>
...
# result clause
SELECT ...
# dataset definition
FROM ...
# query pattern
WHERE {
    ...
}
# query modifiers
ORDER BY ...
```

Adapted from "SPARQL by Example" by Feigenbaum and Prud'hommeaux

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

Today focusing on just simple SELECT

Slide source: Christophe Debruyne

Sample Queries in action

Sample Data for next Example

```
@base <http://foo.bar/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<#Cat>    a rdfs:Class ;     rdfs:label "Cat" .

<#owns>   a rdf:Property ;   rdfs:label "owns" ;
          rdfs:domain foaf:Person ;   rdfs:range <#Cat> .

<#victor>     a <#Cat> ; foaf:name "Victor" .
<#louis>  a <#Cat> ; foaf:name "Louis" .
<#bettina>    a <#Cat> ; foaf:name "Bettina" .

<#chrdebru>   a foaf:Person ;    foaf:name "Christophe Debruyne" ;
              <#owns> <#victor> ; <#owns> <#louis> ;
              <#owns> <#bettina> .
```

Slide source: Christophe Debruyne

Simple SPARQL Queries

```
 PREFIXES omitted
<#Cat> a rdfs:Class ;      rdfs:label "Cat" .

<#owns>       a rdf:Property ;    rdfs:label "owns" ;
              rdfs:domain foaf:Person ; rdfs:range <#Cat> .

<#victor>     a <#Cat> ; foaf:name "Victor" .
<#louis>      a <#Cat> ; foaf:name "Louis" .
<#bettina>    a <#Cat> ; foaf:name "Bettina" .

<#chrdebru>   a foaf:Person ;     foaf:name "Christophe
Debruyne" ;
                    <#owns> <#victor> ; <#owns> <#louis> ;
                    <#owns> <#bettina> .
```

- Finding all cats and their names

```
PREFIX ex: <http://foo.bar/#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?cat ?name
WHERE {  ?cat a ex:Cat.
         ?cat foaf:name ?name.   }
```

  - Variables start with a question mark and can match any term (resource or literal);
  - Triple patterns are just like triples, except that **any of the parts of a triple can be replaced with a variable;**
  - The SELECT result clause returns a table of variables and values that satisfy the query.
  - Recall that “a” is syntactic sugar for “rdf:type”

Slide source: Christophe Debruyne

Simple SPARQL Result

```
PREFIX ex: <http://foo.bar/#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?cat ?name
WHERE {  ?cat a ex:Cat ; foaf:name ?name. }
```

- Finding all cats and their names

```
-----------------------------------------
| cat                          | name     |
=========================================
| <http://foo.bar/#bettina> | "Bettina" |
| <http://foo.bar/#gaston>  | "Gaston"  |
| <http://foo.bar/#victor>  | "Victor"  |
-----------------------------------------
```

Slide source: Christophe Debruyne

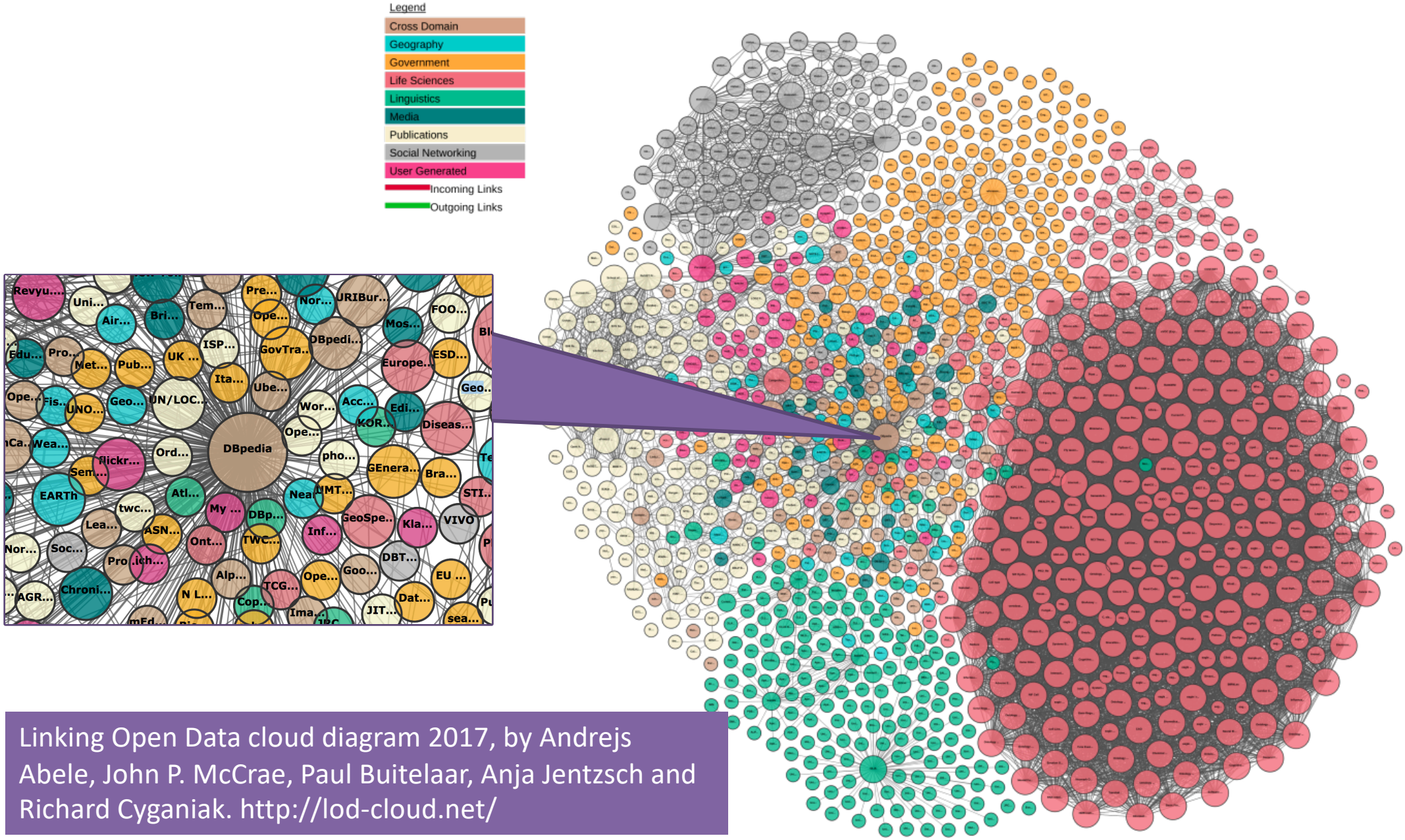
SPARQL Architecture and Endpoints

- 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 (useful for RIA).
  - Certain SPARQL result clauses trigger **RDF** responses.
  - **HTML** often as an XSLT transformation of the XML.
  - **CSV**.

Slide source: Christophe Debruyne

SPARQL Architecture and Endpoints

- For our examples we will use the YASGUI web page (http://yasgui.triply.cc) to access several **live SPARQL endpoints**
  - Most of the example queries will use Dbpedia SPARQL endpoint
- For obvious reasons most well known SPARQL end points **will only allow** you to **query** the dataset, and **will not allow** you to **add to** the dataset or **upload** your own dataset
- We will need to download and use an implementation of a **triplestore** to do development on your own laptop/machine, e.g. GraphDB, Fueski, Virtuoso (will discuss later)
- Also you will need a local triplestore to execute any queries that involve graph insertions or graph constructions



Linking Open Data cloud diagram 2017, by Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch and Richard Cyganiak. http://lod-cloud.net/

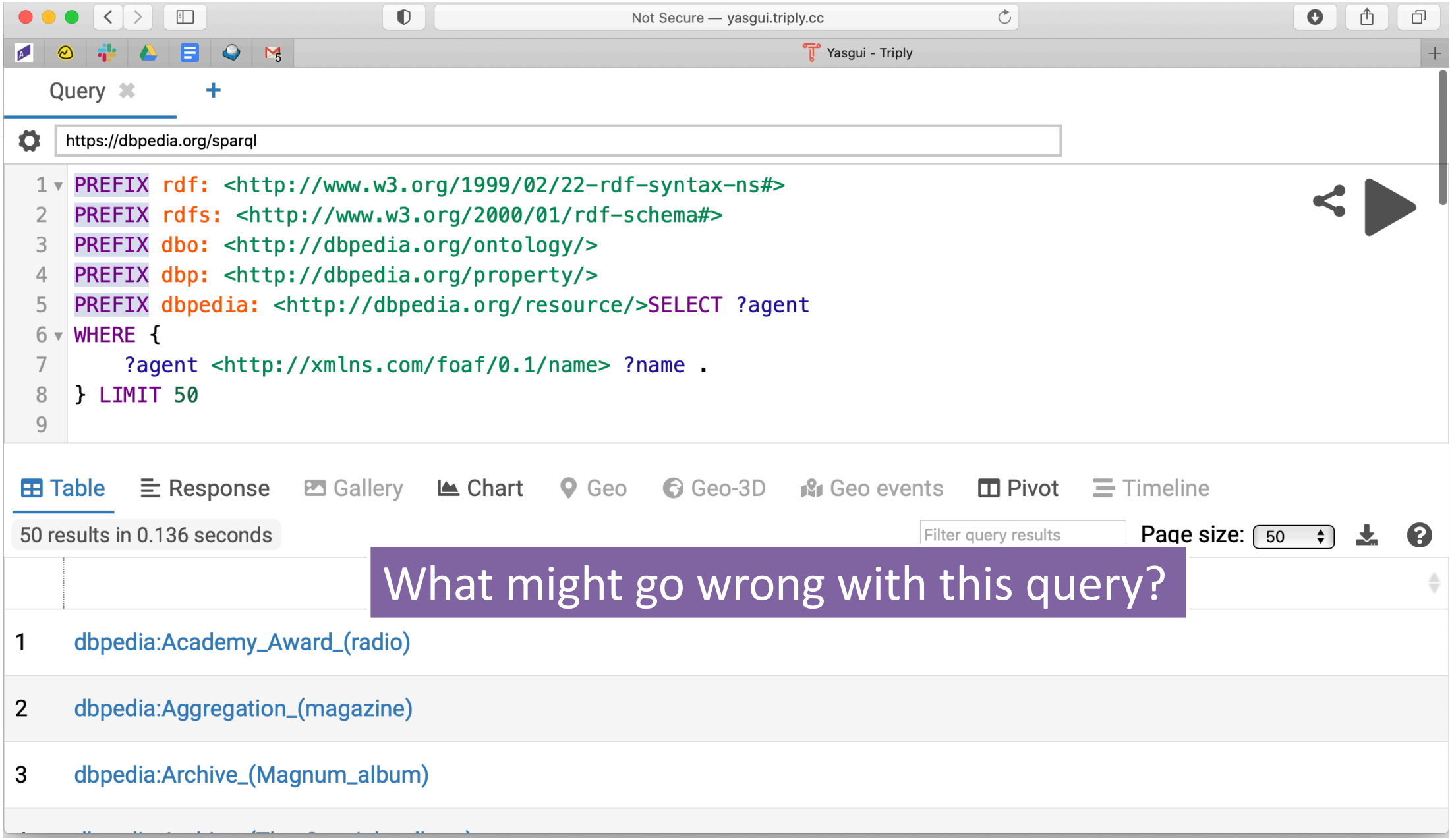
TryItOut 1: Live SPARQL endpoint

- DBpedia is attempting to provide an RDF version of Wikipedia

- Use http://yasgui.triply.cc to "Find me 50 **'things'** with **names** in DBPedia"
- Use https://dbpedia.org/sparql as the endpoint in the top 'Query' box
- Cut and Paste in the query below and execute

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/resource/>
SELECT ?agent
WHERE {
    ?agent <http://xmlns.com/foaf/0.1/name> ?name .
} LIMIT 50
```

Slide source: Christophe Debruyne


Not Secure — yasgui.triply.cc
Yasgui - Triply
Query
https://dbpedia.org/sparql
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3 PREFIX dbo: <http://dbpedia.org/ontology/>
4 PREFIX dbp: <http://dbpedia.org/property/>
5 PREFIX dbpedia: <http://dbpedia.org/resource/>SELECT ?agent
6 WHERE {
7 ?agent <http://xmlns.com/foaf/0.1/name> ?name .
8 } LIMIT 50
9
Table
Response
Gallery
Chart
Geo
Geo-3D
Geo events
Pivot
Timeline
50 results in 0.136 seconds
Filter query results
Page size: 50
What might go wrong with this query?
1 dbpedia:Academy_Award_(radio)
2 dbpedia:Aggregation_(magazine)
3 dbpedia:Archive_(Magnum_album)

Querying SPARQL endpoints

```
SELECT ?agent
WHERE {
     ?agent <http://xmlns.com/foaf/0.1/name> ?name .
} LIMIT 50
```

Likely Duplicates!

Solution Modifiers

- Solution modifiers are useful
  - **`DISTINCT ->`** ensure solutions in sequence are unique
  - **`LIMIT`** → Limit the number of rows returned
  - **`ORDER BY`** → Sorting
  - **`OFFSET`** → Used together with **`LIMIT`** and **`ORDER BY`** for *slicing*, e.g., for paging

  Will only use DISTINCT and LIMIT for moment

```
SELECT DISTINCT ?agent
WHERE {
     ?agent <http://xmlns.com/foaf/0.1/name> ?name .
} LIMIT 50
```

OFFSET

Retrieve the “tenth page” of countries and limit to 5

```
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 ?country_name ?population
WHERE {?country rdfs:label  ?country_name;
prop:populationEstimate ?population.}

ORDER BY ?country_name LIMIT 5 OFFSET 10
```

	country_name	population
1	"Benelux"@en	"28800000"^^xsd:integer
2	"Caribbean Community"@en	"16743693"^^xsd:integer
3	"Central American Integration System"@en	"51152936"^^xsd:integer
4	"Central European Free Trade Agreement"@en	"21907354"^^xsd:integer
5	"Community of Latin American and Caribbean States"@en	"591038580"^^xsd:integer

TryItOut 2: Live SPARQL endpoint

Recall previous query

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
schema#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/resource/>
SELECT ?agent
WHERE {
    ?agent <http://xmlns.com/foaf/0.1/name> ?name .
} LIMIT 50
```

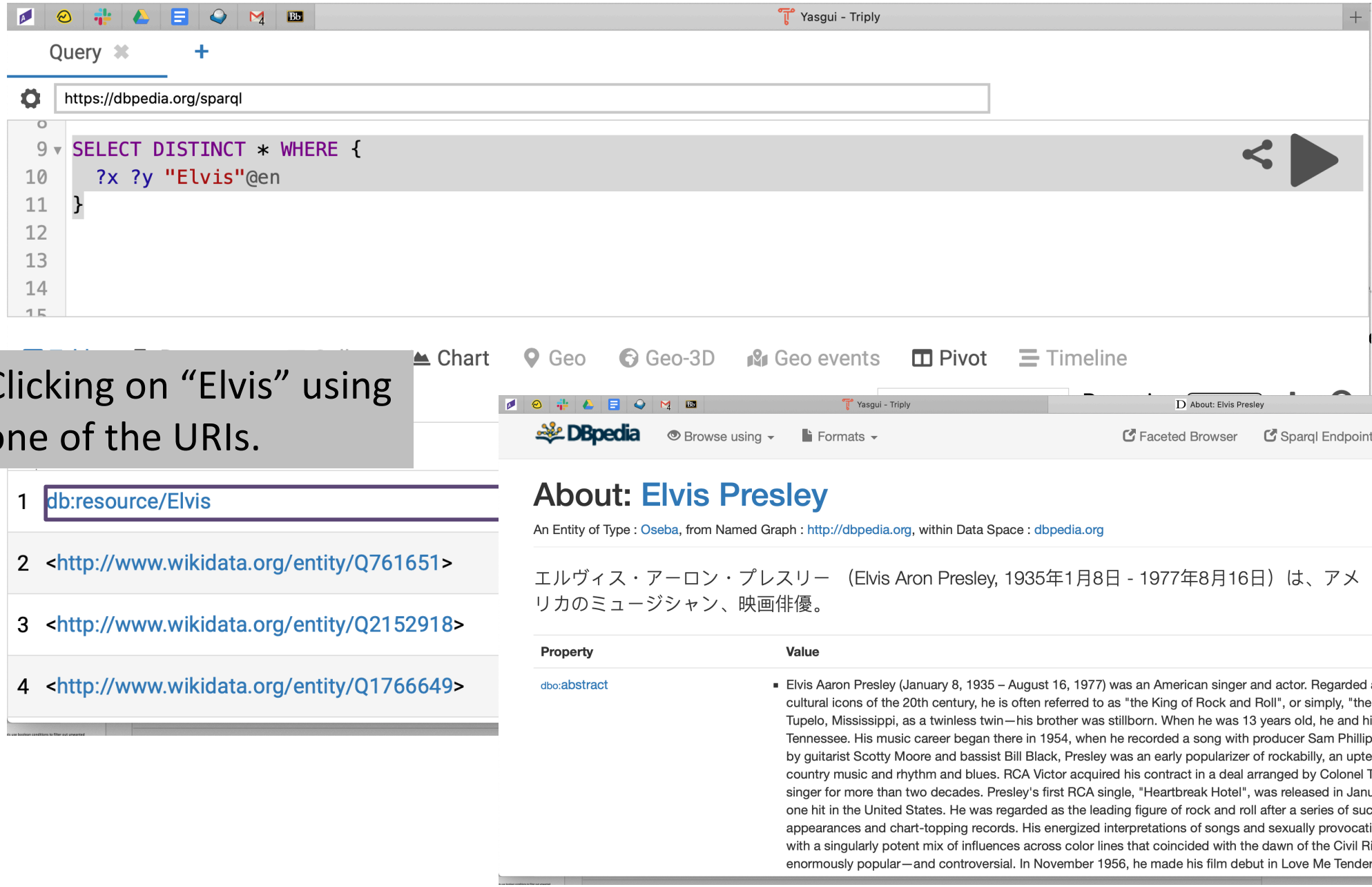
Task: Design a query that

Finds all triples with 'Elvis' as the object

- Tip: do not forget to use the language tag in your query: "Elvis"@en
- Tip: click on the DBpedia URLs to check whether you have found the correct Elvis!

TryOut 2 Solution

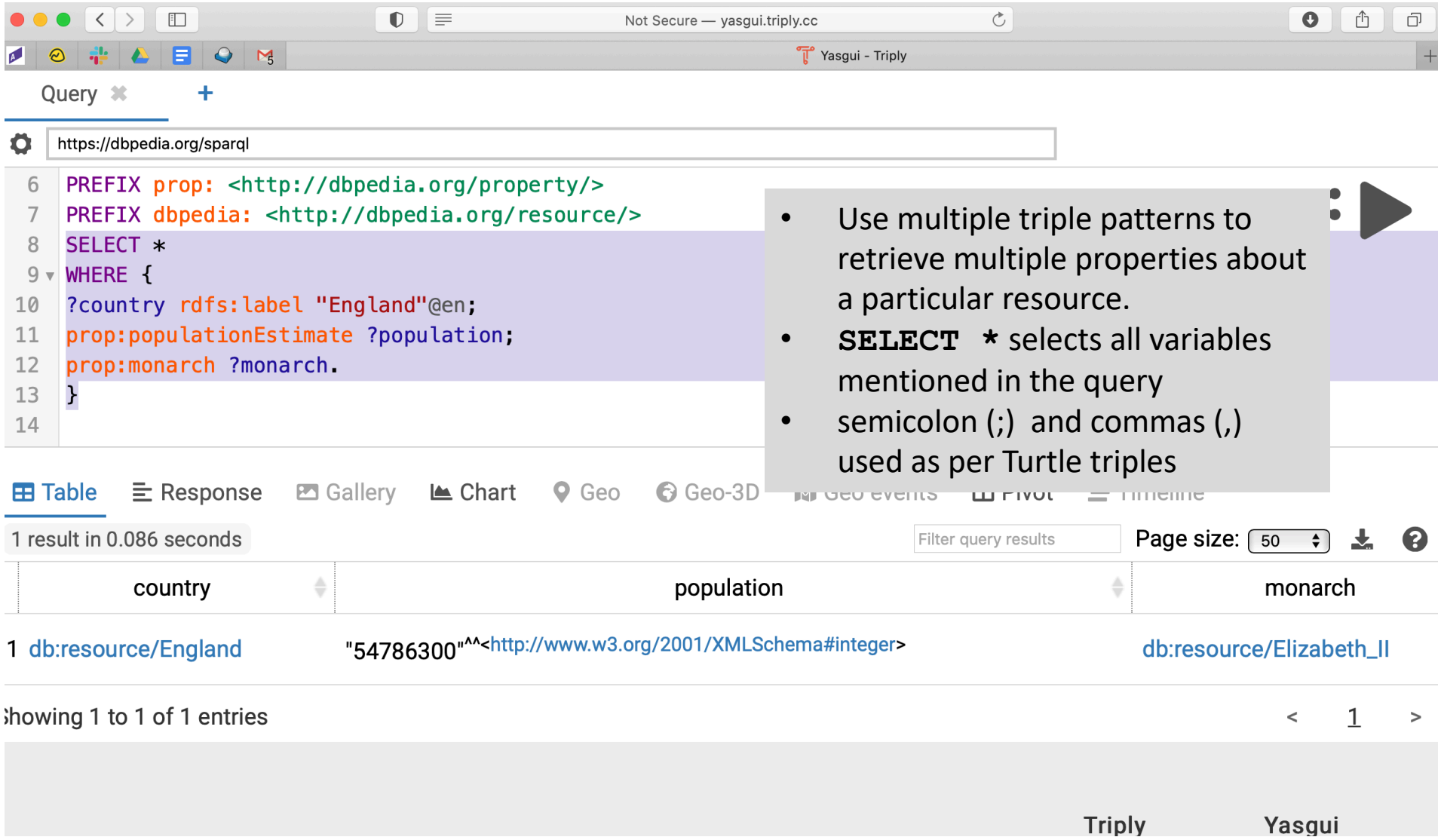
```
SELECT DISTINCT * WHERE {
  ?x ?y "Elvis"@en
}
```



The WHERE CLAUSE

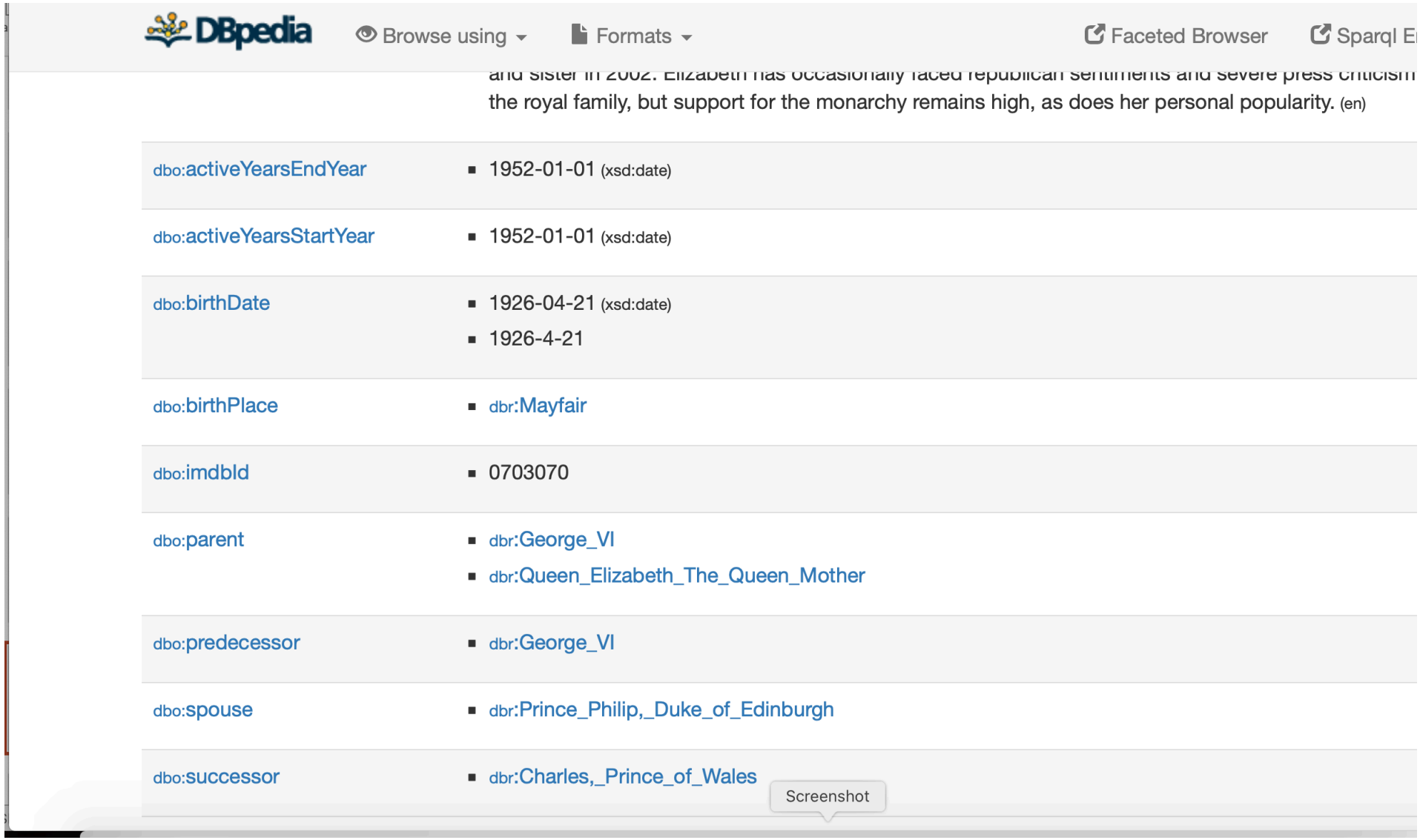
SPARQL QUERY PATTERNS

Example of several query patterns



- Use multiple triple patterns to retrieve multiple properties about a particular resource.
- **`SELECT *`** selects all variables mentioned in the query
- semicolon (;) and commas (,) used as per Turtle triples

When we click (db:resource/Elizabeth_II) in YasGui



TryItOut 3

Task: Design a query that
List people who are born in Dublin and give their names

```
PREFIX dbprop: <http://dbpedia.org/property/>
PREFIX dbo: <http://dbpedia.org/ontology/>
```

Tips:

- dbo: has a predicate **birthPlace**
- Dublin Resource : **<http://dbpedia.org/resource/Dublin>**
- dbprop: has a property **name** that can be used as property that gives name of a person

TryItOut 3 Solution

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

Start of list of results…

1	<http://dbpedia.org/resource/Brendan_Kelly_(actor)>	"Brendan Kelly"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>
2	<http://dbpedia.org/resource/Freeman_Wills_Crofts>	"Freeman Wills Crofts"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>
3	<http://dbpedia.org/resource/Tom_Dunne>	"Tom Dunne"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>
4	<http://dbpedia.org/resource/John_Traynor_(criminal)>	"John Traynor"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>
5	<http://dbpedia.org/resource/Brian_Farrell_(bishop)>	"Brian Farrell"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>
6	<http://dbpedia.org/resource/Peter_Caffrey>	"Peter Caffrey"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>
7	<http://dbpedia.org/resource/William_Desmond_(actor)>	"William Desmond"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>
8	<http://dbpedia.org/resource/Colin_Kenny_(actor)>	"Colin Kenny"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>
9	<http://dbpedia.org/resource/Doreen_Keogh>	"Doreen Keogh"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>

Filtering of query solutions

https://www.w3.org/TR/rdf-sparql-query/

SPARQL 1.0	
Logical	!, &&, ||
Math	+, -, *, /
Comparison:	=, !=, >, <, IN, NOT IN, …
SPARQL Tests	isURI, isBlank, isLiteral, isNumeric, bound
SPARQL Accessors	str, lang, datatype
Other	sameTerm, langMatches, regex, REPLACE

Filtering query solutions

```
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 *
WHERE {
?country rdfs:label ?country_name;
prop:populationEstimate ?population.
FILTER (?population < 7000000)
}
```

- **`FILTER`** constraints use boolean conditions to filter out unwanted query results.

Slide source: Christophe Debruyne

Table | Response | Gallery | Chart | Geo | Geo-3D | Geo

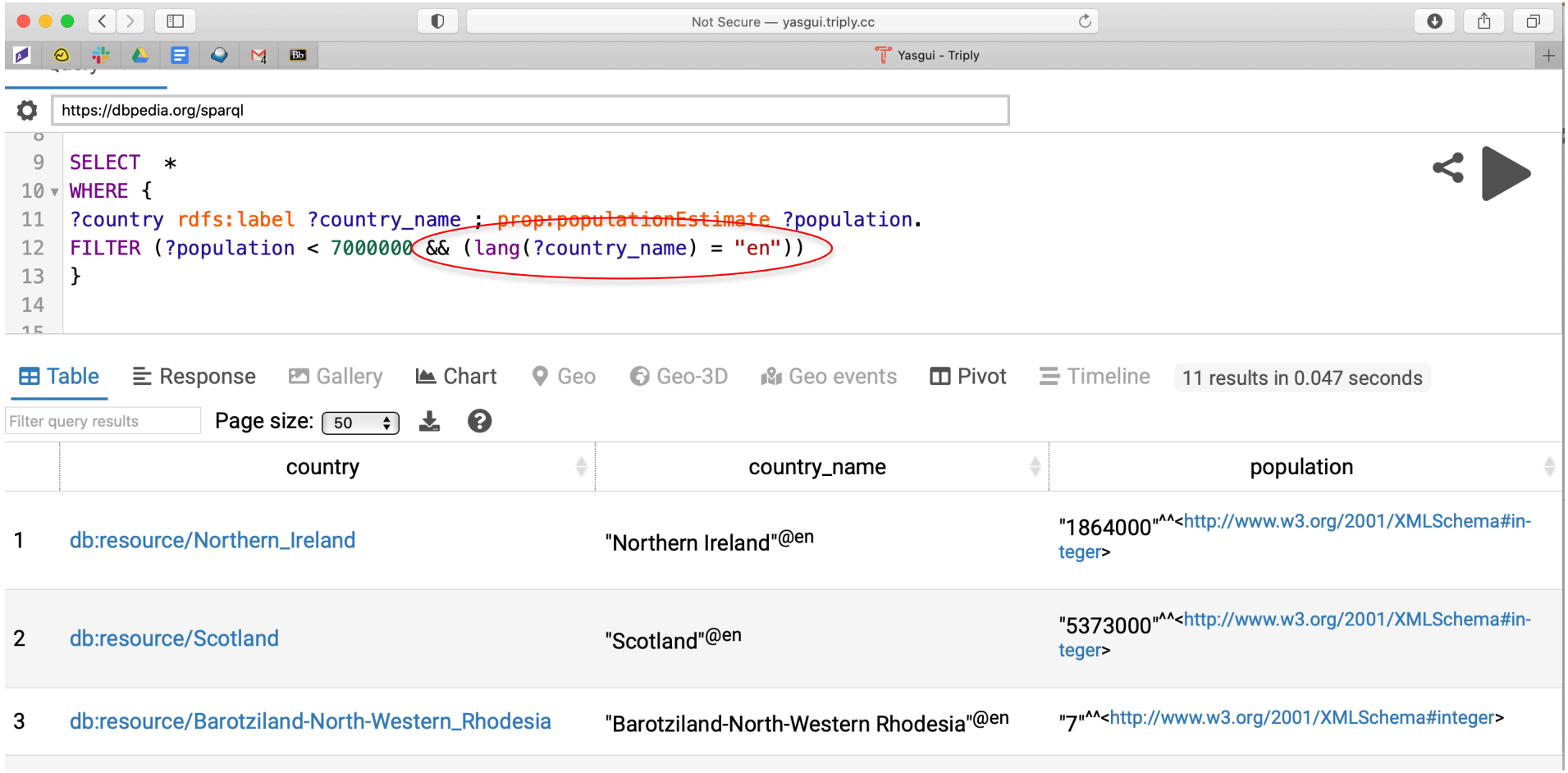
Filter query results | Page size: 50

	country	country_	
1	db:resource/Northern_Ireland	"Northern Ireland"@en	"1864000"^^<http://www.w3.org/2001/XMLSchema#integer>
2	db:resource/Northern_Ireland	"أيرلندا الشمالية"@ar	
3	db:resource/Northern_Ireland	"Nordirland"@de	

What do you notice about the result?

Duplicate results for each language of a label, as distinct triples

Solution.. Add another filter condition



	country	country_name	population
1	db:resource/Northern_Ireland	"Northern Ireland"@en	"1864000"^^<http://www.w3.org/2001/XMLSchema#integer>
2	db:resource/Scotland	"Scotland"@en	"5373000"^^<http://www.w3.org/2001/XMLSchema#integer>
3	db:resource/Barotziland-North-Western_Rhodesia	"Barotziland-North-Western Rhodesia"@en	"7"^^<http://www.w3.org/2001/XMLSchema#integer>

SPARQL Function: Tests

isURI – returns true if variable is a resource/URI

```
FILTER (isURI(?mbox1))
```

isBlank – returns true if variable is a blank node

```
FILTER (isBlank(?mbox2))
```

isLiteral – returns true if variable is a literal value

```
FILTER (isLiteral(?mbox3))
```

bound – returns true if variable is bound to a solution

```
FILTER ( ! bound(?mbox4))
```

SPARQL Functions: Accessors

str – extracts the value of a string or a string representation of a URI

```
<http://example.com/sheep> a rdfs:Resource .
FILTER (str(?s) = 'http://example.com/sheep')
```

lang – extracts a resources language tag, if any

```
eg: employee123 foaf:name "Roberto"@ES .
FILTER (lang(?employee) = 'ES')
```

datatype – extracts the datatype of a literal

```
eg: shoeSize "9.5"^^xsd:float .
FILTER (datatype(?shoeSize) =
<http://www.w3.org/2001/XMLSchema#float>)
```

SPARQL Functions: Other

sameTerm – true if both variables are the same resource

```
FILTER (sameTerm(?mbox1, ?mbox2))
```

langMatches – true if both literals have the same language

```
FILTER (?population > 15000000 &&
langMatches(lang(?country_name), "EN"))
```

Regex – the matching toolbox, possibly slow!

```
FILTER regex(?name, "^ali", "i")
```

(i.e. FILTER regex(?x, "pattern" [, "flags"])

Filtering

https://www.w3.org/TR/sparql11-query/#bind

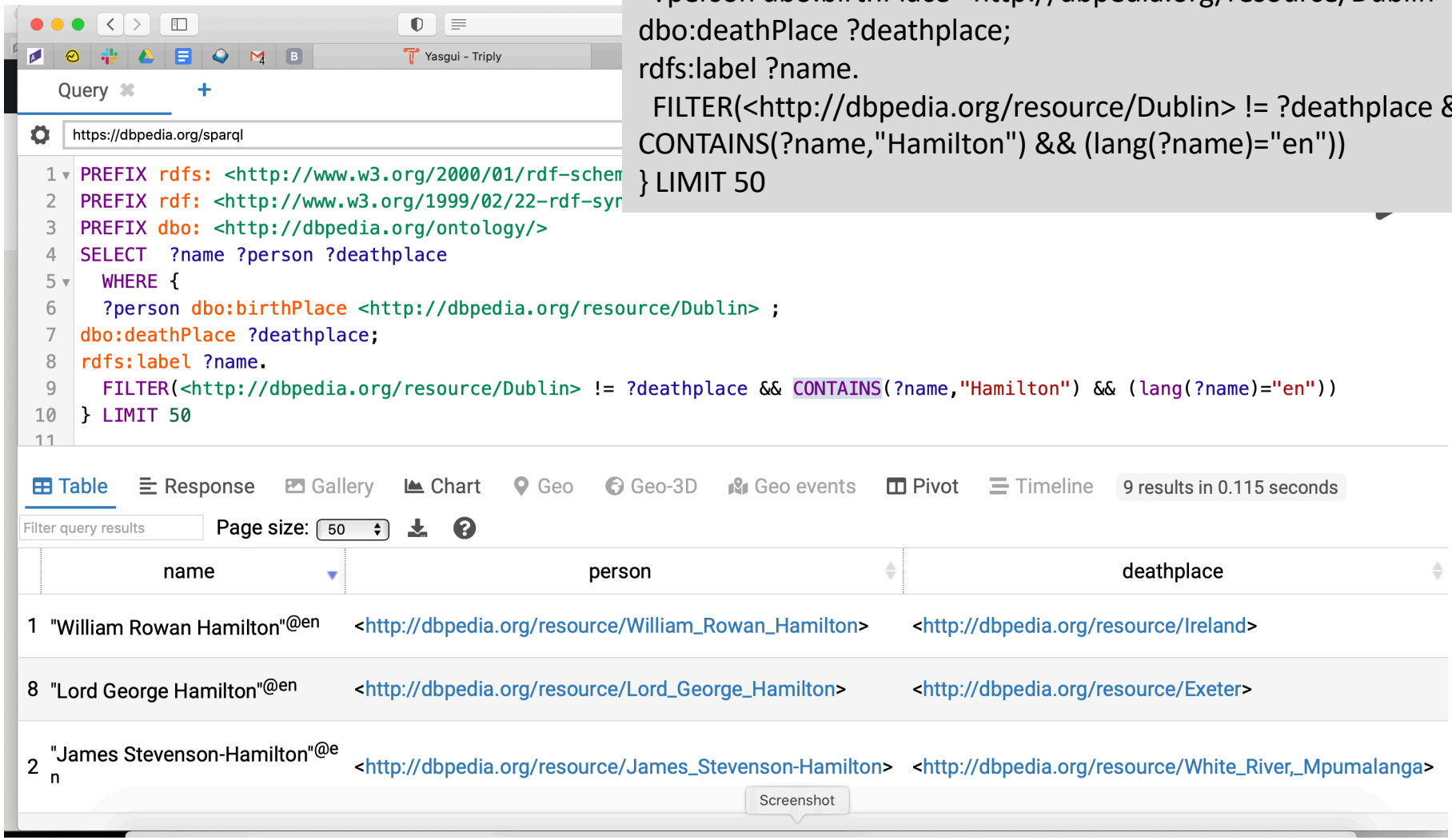
SPARQL 1.1	
Conditionals	IF, COALESCE, EXISTS, NOT EXISTS
Constructors	URI, BNODE, STRLANG, UUID, …
Strings	STRLEN, SUBSTR, UCASE, CONTAINS …
Extra math functions	abs, round, ceil, floor, RAND
Date and Time	now, year, month, day, hours, minutes, …
Hashing	MD5, SHA1, SHA256, SHA384, SHA512

Slide source: Christophe Debruyne

46

Example CONTAINS

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT  ?name ?person ?deathplace
  WHERE {
  ?person dbo:birthPlace <http://dbpedia.org/resource/Dublin> ;
dbo:deathPlace ?deathplace;
rdfs:label ?name.
  FILTER(<http://dbpedia.org/resource/Dublin> != ?deathplace &&
CONTAINS(?name,"Hamilton") && (lang(?name)="en"))
} LIMIT 50
```



TryItOut 4

Recall from earlier

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

Task: Design a query that

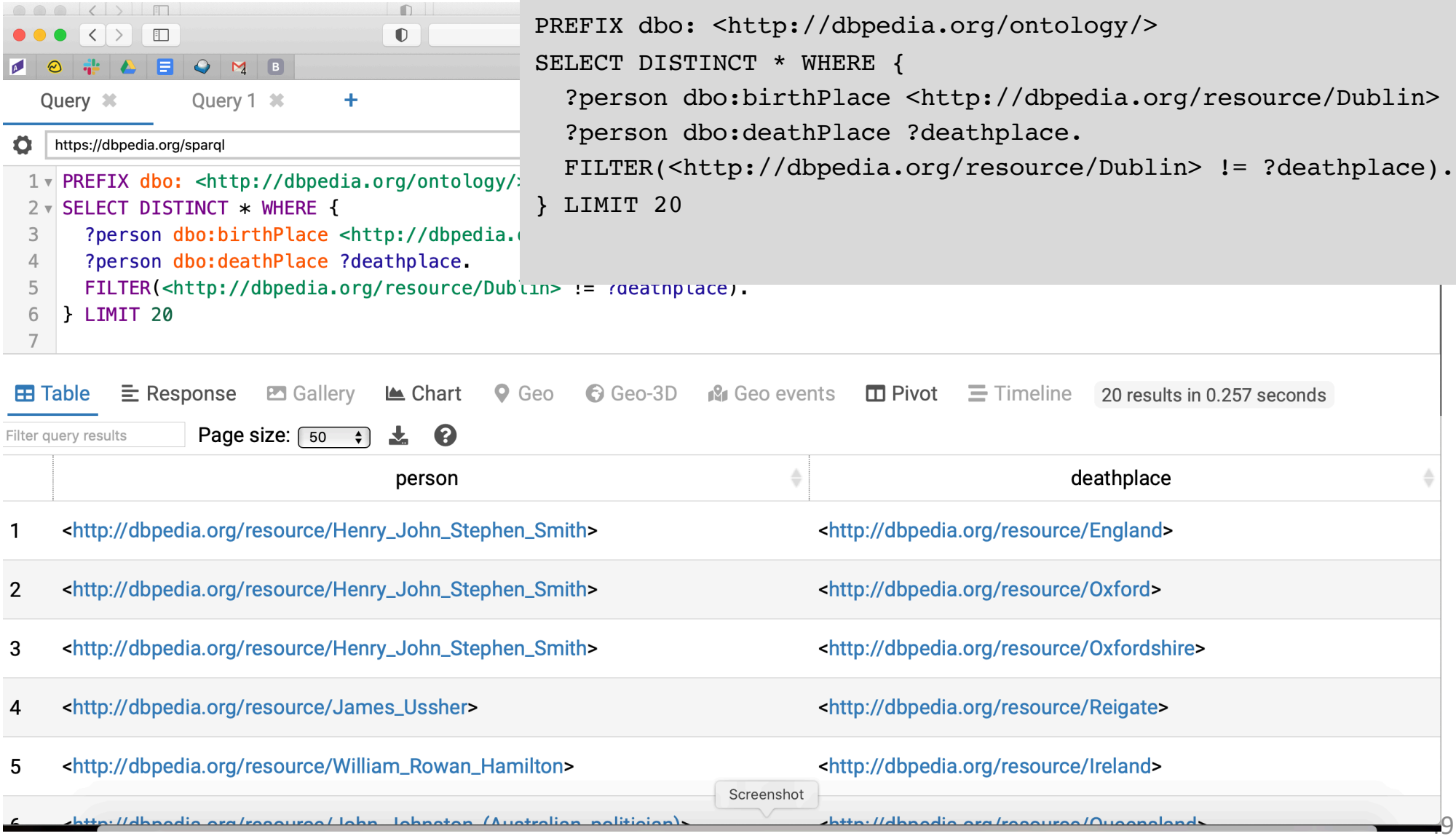
List 20 people (using URI and place of death) who were born in Dublin, but died elsewhere.

Tips:

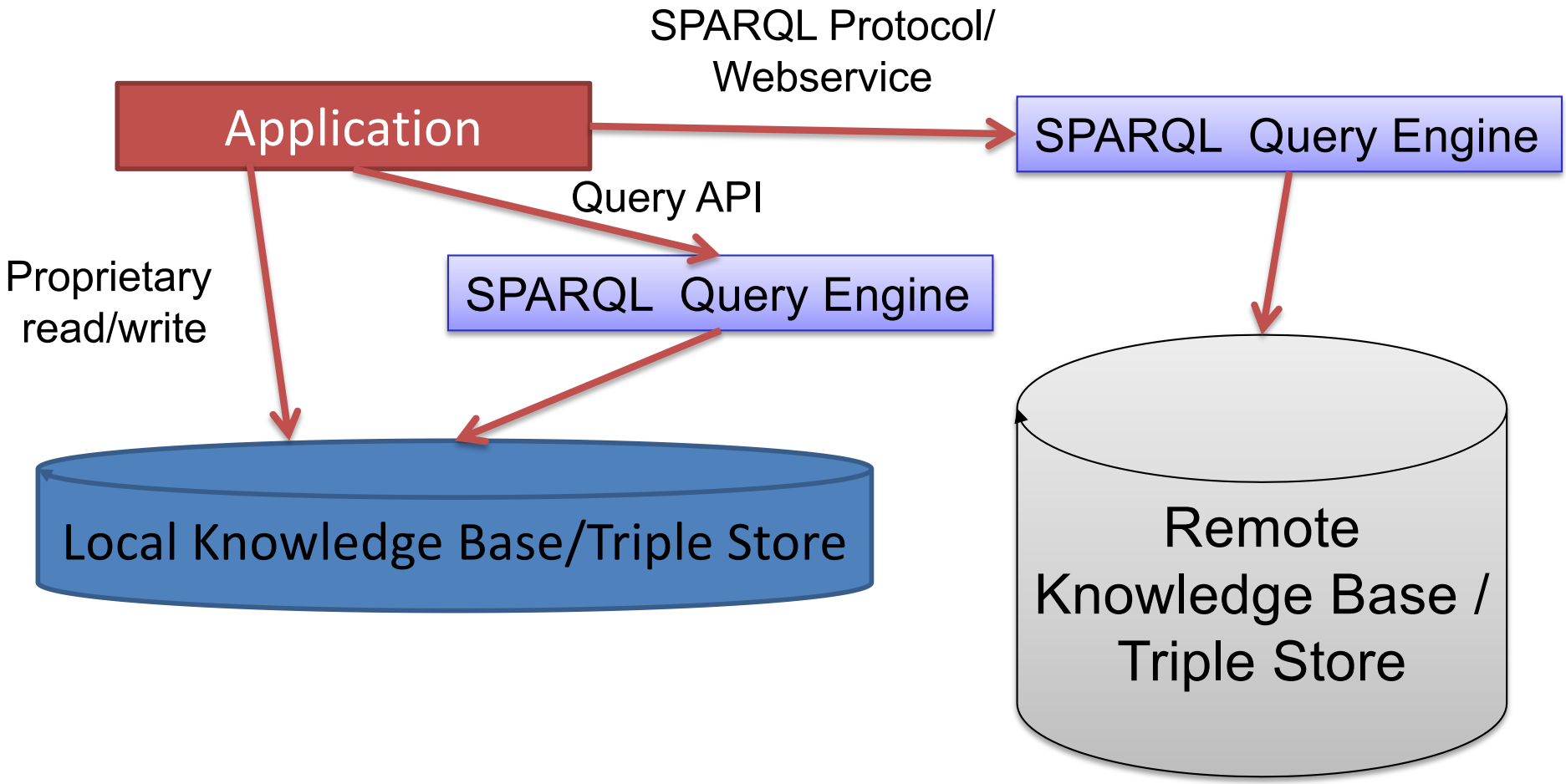
- dbo: has a predicate **birthPlace** and **deathPlace**
- Dublin Resource : **<http://dbpedia.org/resource/Dublin>**

TryItOut 4 Solution

```
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT * WHERE {
  ?person dbo:birthPlace <http://dbpedia.org/resource/Dublin> .
  ?person dbo:deathPlace ?deathplace.
  FILTER(<http://dbpedia.org/resource/Dublin> != ?deathplace).
} LIMIT 20
```



Where SPARQL fits



Slide source: Christophe Debruyne

RDF storage

Triplestore

- Data stored in graphs
- Use SPARQL for querying
- Schema flexibility at the expense of computation
- Expressivity –
  - SPARQL allows queries over property paths (relationships have names, and these can be combined), which cannot be done
- More modern -
  - SPARQL has been designed to be a RESTful service over HTTP so fits into Service Architecture.

Relational Data Base

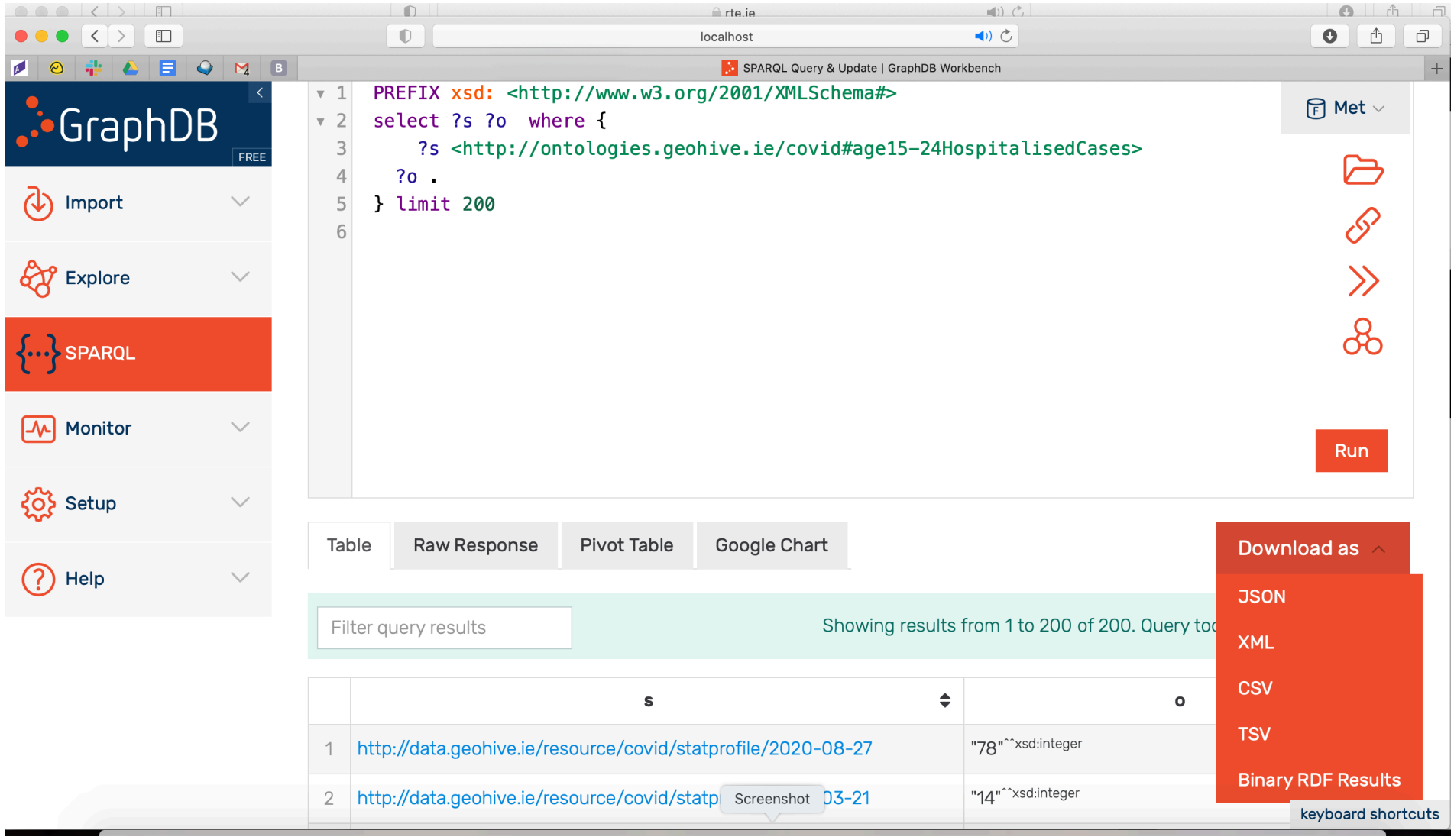
- Data stored in relations (tables)
- Use SQL for querying
- SQL databases used fixed schemas, allowing for optimisation and data integrity at the expense of flexibility
- cost per unit information stored in RDF v's SQL is noticeably higher

RDF storage

Triplestores can be broadly classified in two types **: Native triplestores** and **RDBMS-backed** triplestores.

- **Native triplestores** are those that are implemented from scratch and exploit the RDF data model to efficiently store and access the RDF data.
  - These include: Fuseki Jena, Parliament, Stardog, **GraphDB**, Strabon,, Sesame, 4Store, AllegroGraph, BigData, OWLIM and uRiKa.
- **RDBMS-backed triplestores** are built by adding an RDF specific layer to an existing RDBMS.
  - These include: Jena SDB (no longer actively being developed) and Virtuoso.

https://graphdb.ontotext.com



Self-Directed Exercise Task 6 (SDT6)

1. Download and install a triplestore. (see earlier slide for options)
   For example “GraphDB Free” https://graphdb.ontotext.com
2. Download **SPARQLforStudents.zip** from Resources area of blackboard
3. Try the **TryItOut SPARQL** task by **THIS** Thursday 15th October
4. Try the **Health SPARQL Tutorial** task by Thursday 23rd October
   - Solutions for this tutorial will be made available on 23rd October

For Portfolio

Self-Directed Exercise Task 7 (SDT7)

– before Monday 26th October

1. Rework your Personal Graph (from Self Directed Task 3)
   - Declare a Person class; with subclass of Student
   - Make yourself an instance of the Student Class
   - Add properties with your characteristics (e.g. hobbies etc.)
2. Load your Personal Graph into your triplestore
3. Create a simple SPARQL query on your personal graph
4. Do a screenshot of the triplestore executing SPARQL query and showing Result
5. Post screenshot and the turtle file of personal graph triples into your portfolio

SPARQL Query Debugging Tips

Print interim values of data

Re-write queries to use more variables and add the variables into SELECT

Simplify queries to elemental queries

Get each step working, then re-combine

Verify the data you are looking for is in the store

Eg select all triples:

```
    SELECT ?statement ?pred ?obj
WHERE{
    ?statement ?pred ?obj
    }
```

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/