

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 Protocol And RDF Query Language

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**

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
 - ...

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

Background – W3C standard

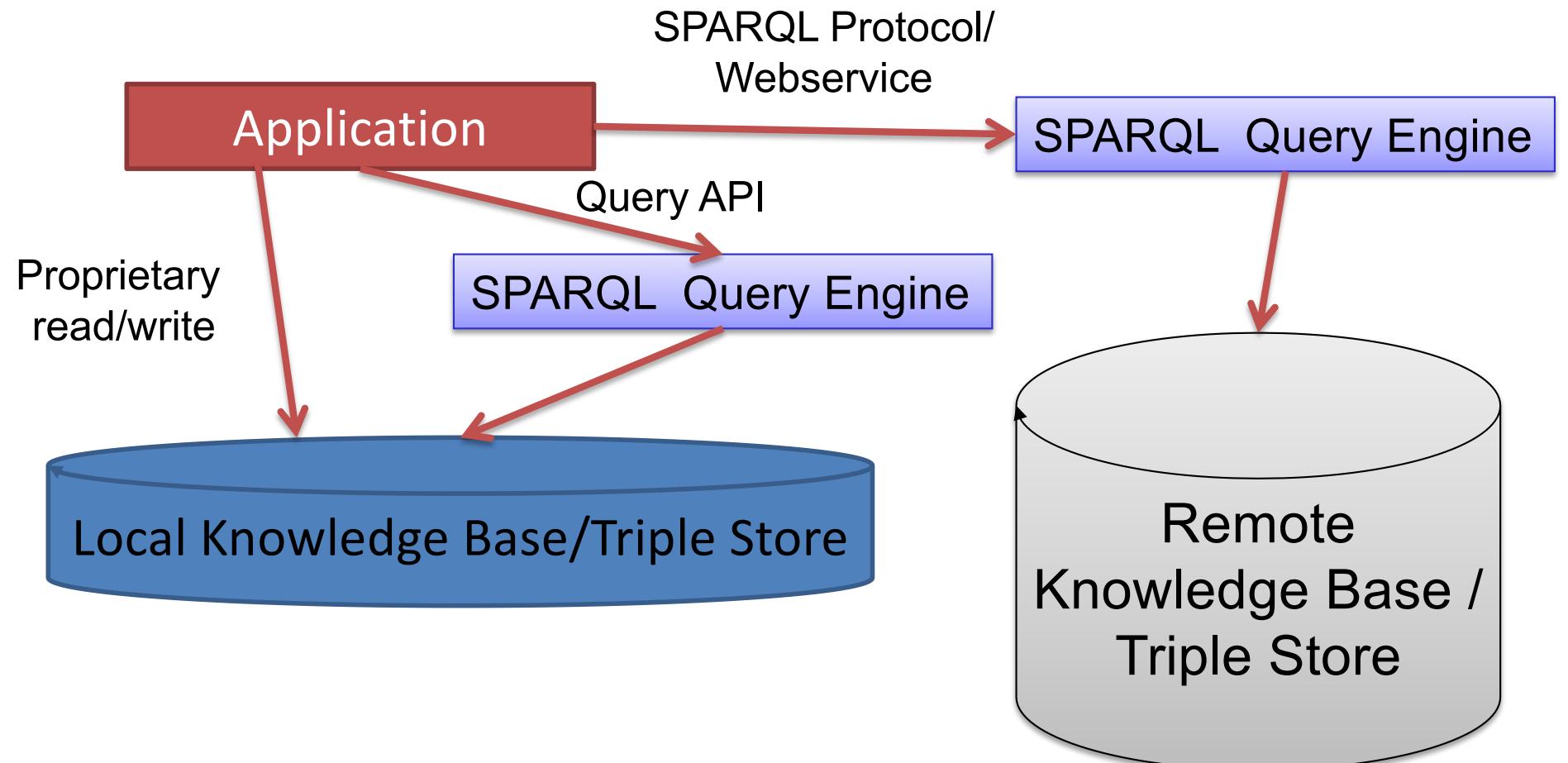
SPARQL 1.0

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

SPARQL 1.1

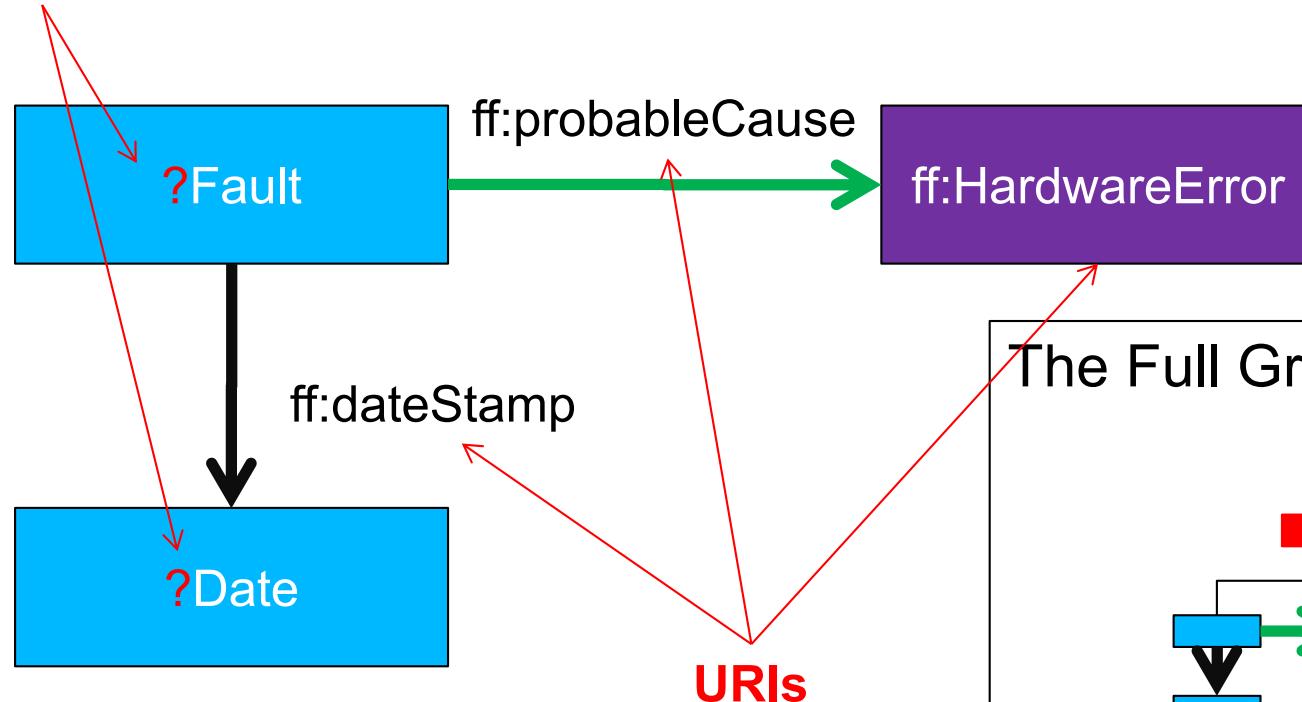
- March 2013
- Several updates
- Additional Features
 - Update (insert, del, modify)
 - Graph Store HTTP protocol
 - Service Descriptions
 - Entailments
 - Basic Federated Query
 - Results in CSV, TSV

Where SPARQL fits

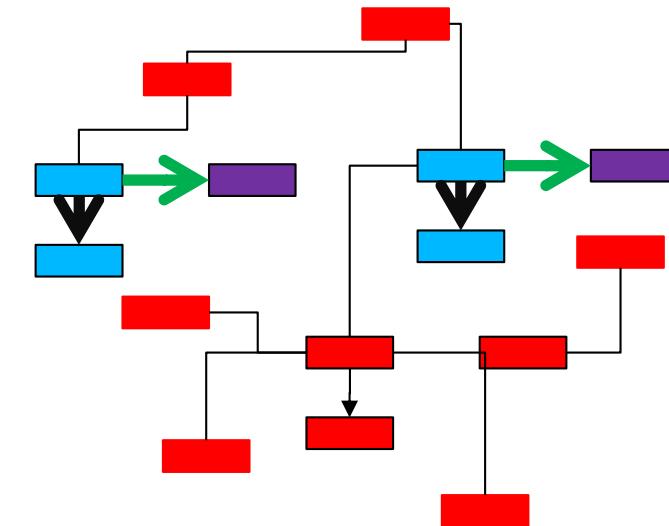


SPARQL based on Graph Matching

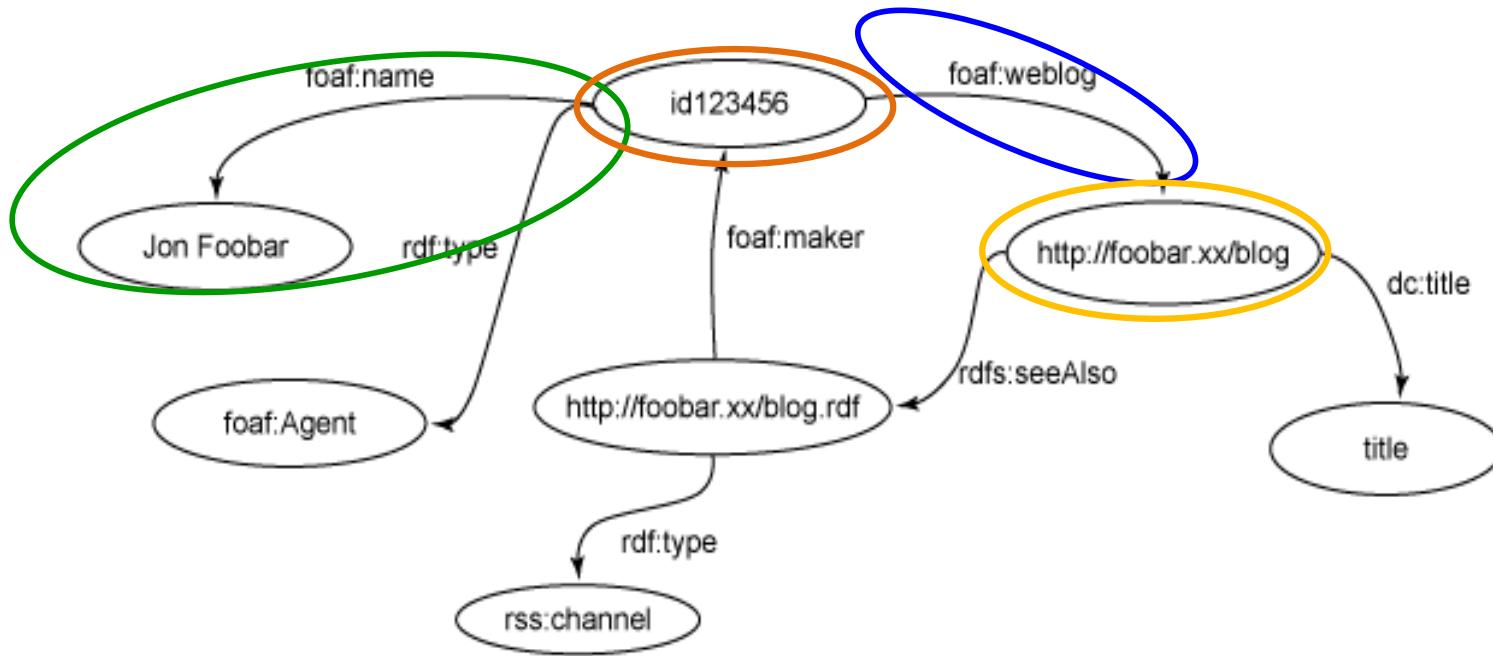
Variables, not URIs,



The Full Graph



Query in action



PREFIX foaf: <<http://xmlns.com/foaf/0.1/>>

<- **NAMESPACE**

SELECT ?url

WHERE {

?contributor foaf:name "Jon Foobar" .

<- **Triple patterns using
Turtle syntax**

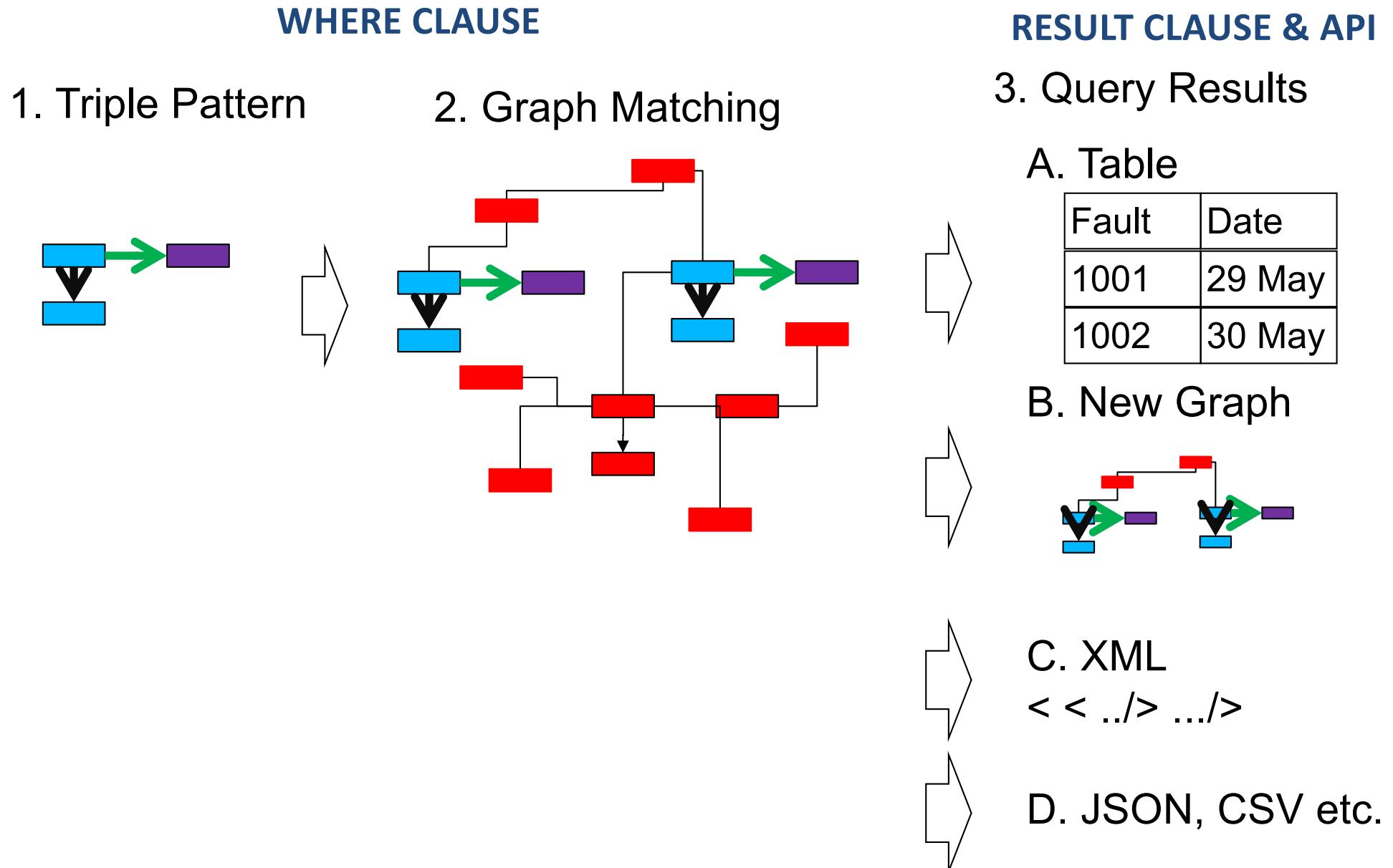
?contributor foaf:weblog ?url .

}

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



Structure of a SPARQL Query

Namespace shortened for brevity

A SPARQL query comprises, in order:

- **Prefix declarations** for URI shorthands

```
# 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

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> .
```



Simple SPARQL Queries

- 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. }
```

```
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> .
```

- 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”

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

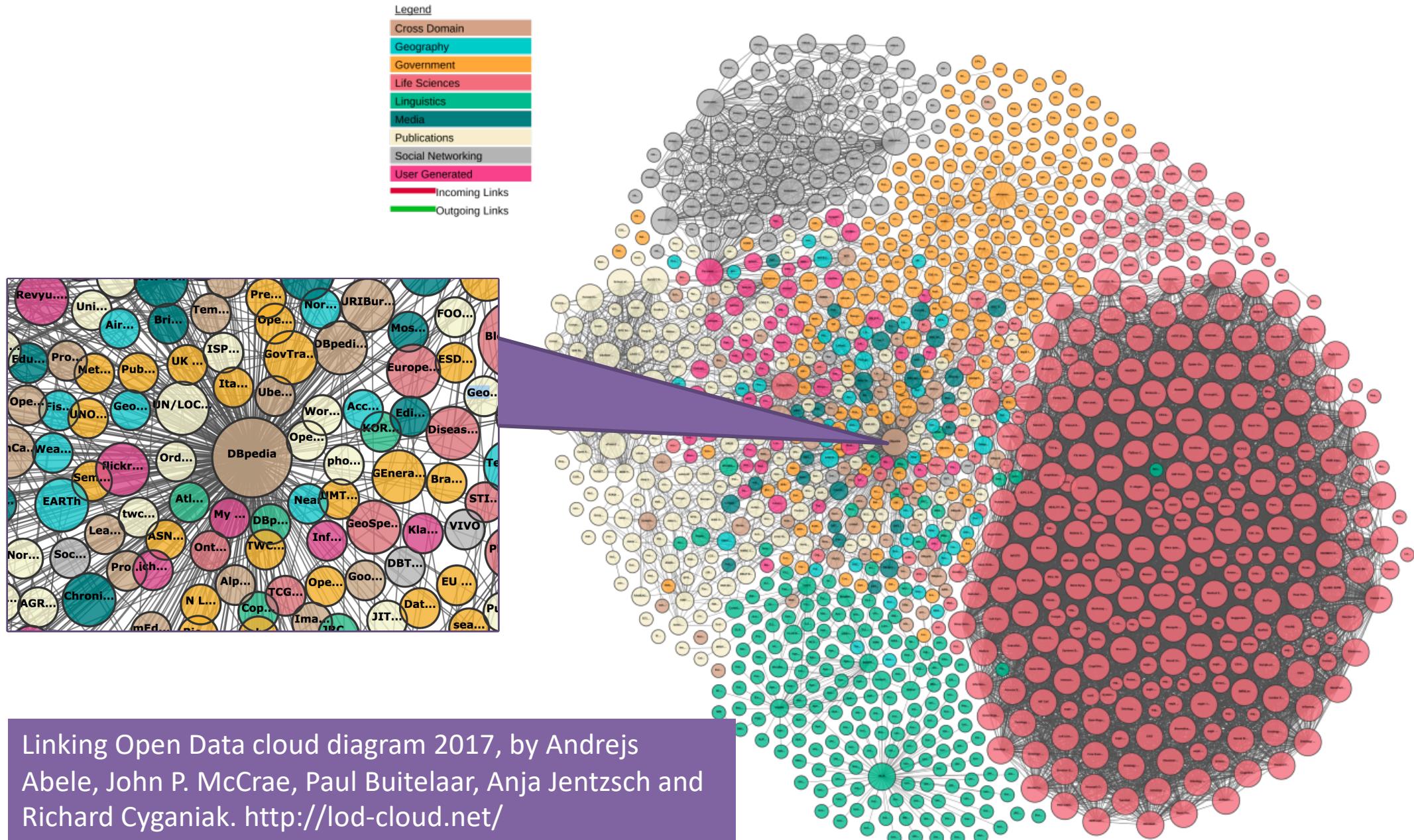
cat	name
<http://foo.bar/#bettina>	"Bettina"
<http://foo.bar/#gaston>	"Gaston"
<http://foo.bar/#victor>	"Victor"

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**.

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.triplay.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
```

Not Secure — yasgui.triply.cc

Yasgui - Triply

Query X +

⚙️ <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

Share Run

Table Response Gallery Chart Geo Geo-3D Geo events Pivot Timeline

50 results in 0.136 seconds

Filter query results

Page size: 50 Download ?

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
```

The screenshot shows a web browser window with the URL `dbpedia.org/sparql?default-graph-uri=http%3A%2F%2Fdb...`. The page displays a list of agent names under the heading "agent". The list includes:

- http://dbpedia.org/resource/!Action_Pact
- http://dbpedia.org/resource/%22Solidarity%22_Szczecin-Goleni%C3%B3w_Airport
- http://dbpedia.org/resource/%22The_Take_Over,_the_Breaks_Over%22
- http://dbpedia.org/resource/%C3%81g%C3%A6tis_byrjun
- http://dbpedia.org/resource/%C3%81kos_R%C3%A1thonyi
- http://dbpedia.org/resource/%C3%81lvaro_Arz%C3%BA
- http://dbpedia.org/resource/%C3%81lvaro_Arz%C3%BA
- http://dbpedia.org/resource/%C3%81lvaro_Colom
- http://dbpedia.org/resource/%C3%81lvaro_Colom
- http://dbpedia.org/resource/%C3%81lvaro_Rafael_Gonz%C3%A1lez

A purple arrow points from the query text above the browser window to the browser window itself. A purple box in the bottom right corner of the browser window contains the text "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 screenshot shows the TryOut 2 interface. At the top, there's a toolbar with various icons. Below it is a header bar with a gear icon, a URL field containing "https://dbpedia.org/sparql", and a user name "Yasgui - Triply". The main area is a code editor with a "Query" tab selected. The query text is:

```
9 ▼ SELECT DISTINCT * WHERE {  
10   ?x ?y "Elvis"@en  
11 }  
12  
13  
14  
15
```

To the right of the code editor is a button bar with a share icon and a play icon. Below the code editor is a navigation bar with tabs: Chart, Geo, Geo-3D, Geo events, Pivot, and Timeline. The "Geo" tab is currently active.

Clicking on “Elvis” using one of the URLs.

- 1 [db:resource/Elvis](#)
- 2 <<http://www.wikidata.org/entity/Q761651>>
- 3 <<http://www.wikidata.org/entity/Q2152918>>
- 4 <<http://www.wikidata.org/entity/Q1766649>>

The screenshot shows the DBpedia Faceted Browser interface. At the top, there's a header bar with a DBpedia logo, a "Browse using" dropdown, a "Formats" dropdown, and links for "Faceted Browser" and "Sparql Endpoint". The main content area has a title "About: Elvis Presley" and a subtitle "An Entity of Type : Oseba, from Named Graph : http://dbpedia.org, within Data Space : dbpedia.org". Below this is a summary text in Japanese: "エルヴィス・アーロン・プレスリー (Elvis Aron Presley, 1935年1月8日 - 1977年8月16日) は、アメリカのミュージシャン、映画俳優。". A table below shows properties and values for the entity:

Property	Value
dbo:abstract	<p>Elvis Aaron Presley (January 8, 1935 – August 16, 1977) was an American singer and actor. Regarded as one cultural icons of the 20th century, he is often referred to as "the King of Rock and Roll", or simply, "the King". Tupelo, Mississippi, as a twinless twin—his brother was stillborn. When he was 13 years old, he and his family moved to Memphis, Tennessee. His music career began there in 1954, when he recorded a song with producer Sam Phillips at Sun Records. After being discovered by guitarist Scotty Moore and bassist Bill Black, Presley was an early popularizer of rockabilly, an uptempo, blues-influenced genre that combined elements of country music and rhythm and blues. RCA Victor acquired his contract in a deal arranged by Colonel Tom Parker. Presley became a major figure in the rock and roll genre, singing for more than two decades. Presley's first RCA single, "Heartbreak Hotel", was released in January 1956 and became a massive hit in the United States. He was regarded as the leading figure of rock and roll after a series of successful appearances and chart-topping records. His energized interpretations of songs and sexually provocative performances with a singularly potent mix of influences across color lines that coincided with the dawn of the Civil Rights Movement made him an enormously popular—and controversial. In November 1956, he made his film debut in <i>Love Me Tender</i>. In 1957, he starred in his first feature film, <i>King Creole</i>, which was a critical and commercial success.</p>

The WHERE CLAUSE

SPARQL QUERY PATTERNS

Example of several query patterns

The screenshot shows the Yasgui interface for querying a triple store. The top navigation bar includes icons for file operations, a shield for security, and tabs for 'Query' and 'Yasgui - Triply'. The main area has a toolbar with various icons. A query editor window is open, showing a SPARQL query:

```
6 PREFIX prop: <http://dbpedia.org/property/>
7 PREFIX dbpedia: <http://dbpedia.org/resource/>
8 SELECT *
9 WHERE {
10 ?country rdfs:label "England"@en;
11 prop:populationEstimate ?population;
12 prop:monarch ?monarch.
13 }
14 }
```

To the right of the query, a callout box highlights features of the query:

- 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

Below the query editor, there are tabs for 'Table', 'Response', 'Gallery', 'Chart', 'Geo', 'Geo-3D', 'Geo events', 'Pivot', and 'Timeline'. The 'Table' tab is selected, showing the results of the query:

country	population	monarch
1 db:resource/England	"54786300"^^<http://www.w3.org/2001/XMLSchema#integer>	db:resource/Elizabeth_II

Below the table, it says 'Showing 1 to 1 of 1 entries'. At the bottom right, there are buttons for 'Triply' and 'Yasgui'.

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

The screenshot shows the DBpedia YasGui interface for the entity [db:resource/Elizabeth_II](#). The top navigation bar includes links for 'Browse using' (with a dropdown arrow), 'Formats' (with a dropdown arrow), 'Faceted Browser', and 'Sparql E'. Below the navigation, there is a summary text about Queen Elizabeth II's reign and popularity. The main content area lists various properties and their corresponding values:

dbo:activeYearsEndYear	▪ 1952-01-01 (xsd:date)
dbo:activeYearsStartYear	▪ 1952-01-01 (xsd:date)
dbo:birthDate	▪ 1926-04-21 (xsd:date) ▪ 1926-4-21
dbo:birthPlace	▪ dbr:Mayfair
dbo:imdbId	▪ 0703070
dbo:parent	▪ dbr:George_VI ▪ dbr:Queen_Elizabeth_The_Queen_Mother
dbo:predecessor	▪ dbr:George_VI
dbo:spouse	▪ dbr:Prince_Philip,_Duke_of_Edinburgh
dbo:successor	▪ dbr:Charles,_Prince_of_Wales

A 'Screenshot' button is located at the bottom right of the interface.

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/resource/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX yago: <http://dbpedia.org/class/yago/>
PREFIX prop: <http://dbpedia.org/resource/>
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.

Not Secure — yasgui.triply.cc

Yasgui - Triply

```
8
9 SELECT *
10 WHERE {
11 ?country rdfs:label ?country_name; prop:populationEstimate ?population.
12 FILTER (?population < 7000000)
13 }
14
15
16
```

Table Response Gallery Chart Geo Geo-3D Geo

Filter query results Page size: 50 [Download](#) [?](#)

	country	country
1	db:resource/Northern_Ireland	"Northern Ireland"@en
2	db:resource/Northern_Ireland	"أيرلندا الشمالية"@ar
3	db:resource/Northern_Ireland	"Nordirland"@de

What do you notice about the result?

"1864000"^{^^}<http://www.w3.org/2001/XMLSchema#integer>

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

Solution.. Add another filter condition

The screenshot shows a web browser window titled "Not Secure — yasgui.triplay.cc" with the URL "https://dbpedia.org/sparql". The query editor contains the following SPARQL code:

```
9 SELECT *
10 WHERE {
11   ?country rdfs:label ?country_name ; prop:populationEstimate ?population.
12   FILTER (?population < 7000000 && (lang(?country_name) = "en"))
13 }
14
15 }
```

A red oval highlights the filter condition in line 12: `?population < 7000000 && (lang(?country_name) = "en")`. The results table below shows three rows of data:

	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

Example CONTAINS

The screenshot shows the Yasgui - Triply interface. At the top, there is a code editor window containing a SPARQL query. Below it is a results table.

SPARQL Query:

```
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
```

Results Table:

	name	person	deathplace
1	"William Rowan Hamilton"@en	<http://dbpedia.org/resource/William_Rowan_Hamilton>	<http://dbpedia.org/resource/Ireland>
8	"Lord George Hamilton"@en	<http://dbpedia.org/resource/Lord_George_Hamilton>	<http://dbpedia.org/resource/Exeter>
2	"James Stevenson-Hamilton"@e n	<http://dbpedia.org/resource/James_Stevenson-Hamilton>	<http://dbpedia.org/resource/White_River,_Mpumalanga>

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

The screenshot shows a Sparql query interface with the following details:

Query Editor:

```
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
```

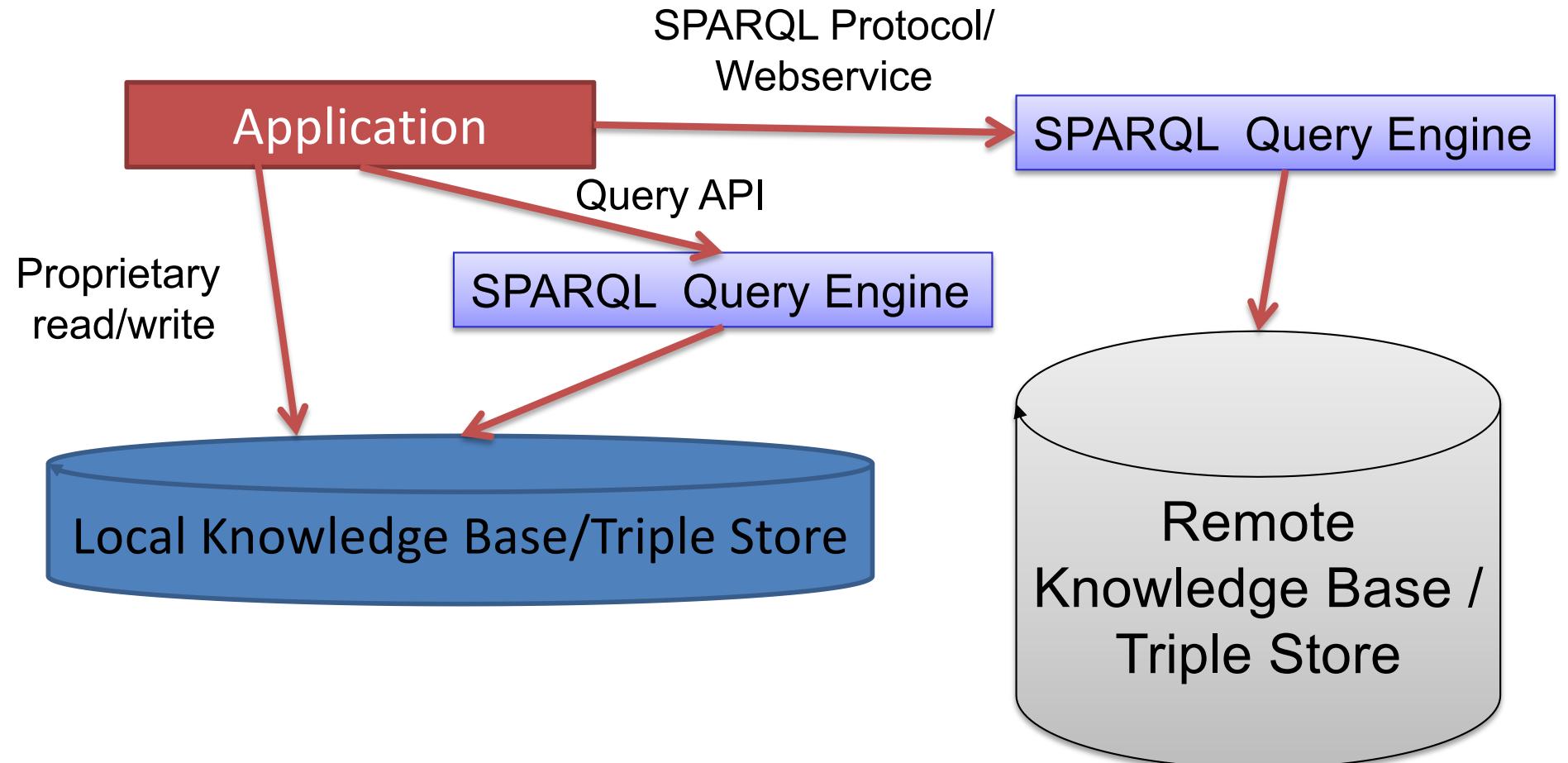
Results Table:

	person	deathplace
1	<http://dbpedia.org/resource/Henry_John_Stephen_Smith>	<http://dbpedia.org/resource/England>
2	<http://dbpedia.org/resource/Henry_John_Stephen_Smith>	<http://dbpedia.org/resource/Oxford>
3	<http://dbpedia.org/resource/Henry_John_Stephen_Smith>	<http://dbpedia.org/resource/Oxfordshire>
4	<http://dbpedia.org/resource/James_Ussher>	<http://dbpedia.org/resource/Reigate>
5	<http://dbpedia.org/resource/William_Rowan_Hamilton>	<http://dbpedia.org/resource/Ireland>
6	<http://dbpedia.org/resource/John_Johnston_(Australian_politician)>	<http://dbpedia.org/resource/Queensland>

Interface Elements:

- Toolbar: Back, Forward, Stop, Refresh, New, Open, Save, Copy, Paste, Find, Help.
- Address Bar: https://dbpedia.org/sparql
- Buttons: Table, Response, Gallery, Chart, Geo, Geo-3D, Geo events, Pivot, Timeline.
- Text: 20 results in 0.257 seconds
- Filter: Filter query results, Page size: 50, Download, Help.

Where SPARQL fits



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>

The screenshot shows the GraphDB Workbench interface running in a web browser. The left sidebar contains navigation links: Import, Explore, SPARQL (highlighted in orange), Monitor, Setup, and Help. The main area displays a SPARQL query:

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
select ?s ?o where {
  ?s <http://ontologies.geohive.ie/covid#age15–24HospitalisedCases>
  ?o .
} limit 200
```

The 'Run' button is visible on the right. Below the query, there are four tabs: Table (selected), Raw Response, Pivot Table, and Google Chart. A 'Filter query results' input field is present. The results table shows two rows:

	s	o
1	http://data.geohive.ie/resource/covid/statprofile/2020-08-27	"78"^^xsd:integer
2	http://data.geohive.ie/resource/covid/statp	"14"^^xsd:integer

A 'Download as' dropdown menu on the right offers options: JSON, XML, CSV, TSV, and Binary RDF Results. A 'Screenshot 03-21' button is located near the bottom of the results table. The bottom right corner includes 'keyboard shortcuts'.

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  
}
```

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/>