Artificial Intelligence Fundamentals

Master in Artificial Intelligence

Practicum 2: Ontologies

Academic year 2025/26

Practice P2.2: Use of SPARQL to query semantic data

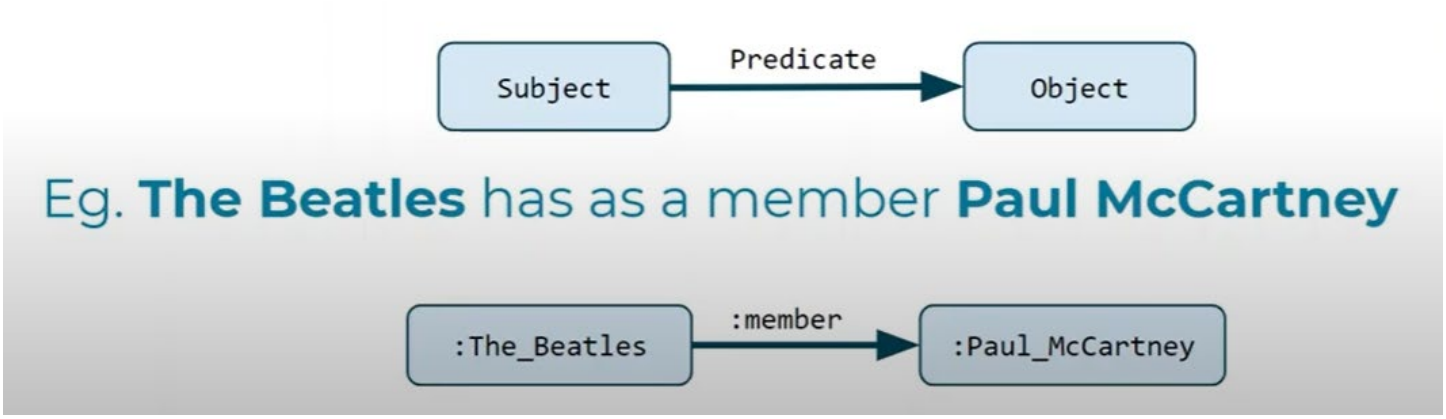
During this practise you will learn how to query **knowledge datasets** (ontology-format data).

- **SPARQL** is a protocol and query language for accessing the **RDF** developed by "W3C RDF Data Access Working Group". The RDF graph is a group of triples.
- It can carry out all the analytics that SQL can perform. Also, it can be utilized for semantic analysis and analysing relationships. SQL to query relational databases, we use SPARQL to query graph databases (sometimes referred to as a triple-stores). As a query language, SPARQL is "data-oriented" in that it only queries the information held in the models.
- Graph databases store data slightly differently than relational databases. Instead of storing data in rows and columns, graph databases store facts. Facts are represented as three values, **subject**, **predicate**, and **object** (sometimes referred to as **subject**, **property**, and **value**) which read a bit like sentences:

  John (*subject*) had (*predicate*) a name (*object*).

- Resources are represented with **URIs**, which can be abbreviated as **prefixed** names. **IRI** = nodes and edges with a unique identifier.
- **Objects** can be literals: strings, integers, booleans, etc. (nodes representing values)
- SPARQL keywords are **case-insensitive** so one can use lowercase keywords like *select* instead of *SELECT*. We recommend consistency.

https://www.youtube.com/watch?v=bDxclRhDb-o



- Literals are written in quotes followed by their datatype IRI

```
"1963-03-22"^^xsd:date
```

```
"1963-03-22T21:44:00Z"^^xsd:dateTime
```

- Datatype can be omitted for strings:

```
"The Beatles"
```

```
"The Beatles"^^xsd:string
```

- Datatype and quotes can be omitted for some datatypes

```
125
```
integer

```
3.14
```
decimal

```
3.2E4
```
double

```
true
```
boolean

SPARQL Architecture & Endpoints

- SPARQL queries are executed against **RDF datasets** (RDF graphs).
- A **SPARQL endpoint** accepts queries and returns results via HTTP.
    - *Generic* endpoints will query any Web-accessible RDF data
    - *Specific* endpoints are hardwired to query against particular datasets
- The **results** of SPARQL queries can be obtained in different formats: XML, JSON, RDF, HTML.

Structure of a SPARQL Query

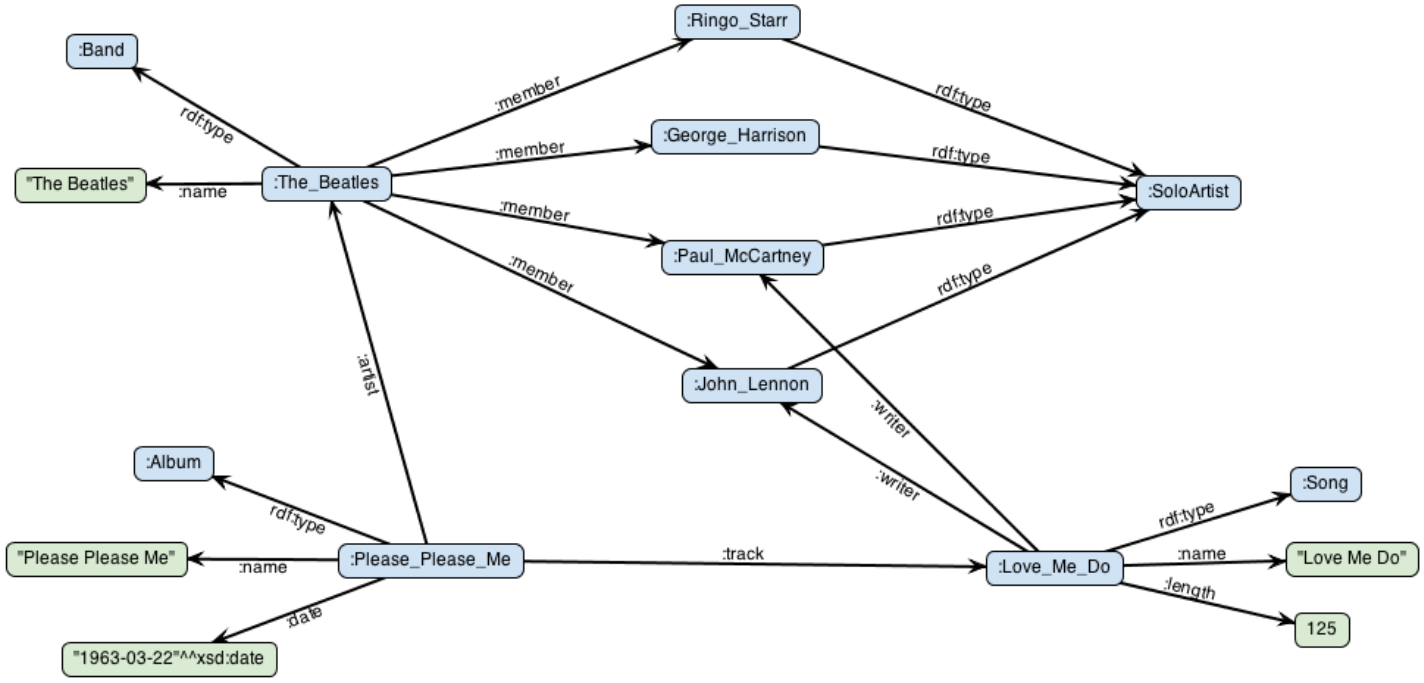
Details about SPARQL: https://www.w3.org/2009/Talks/0615-qbe/

SPARQL query

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

```
# prefix declarations

PREFIX foo: <http://example.com/resources/>

...

# dataset definition

FROM ...

# result clause

SELECT ...

# query pattern

WHERE {

   ...

}

# query modifiers

ORDER BY ...
```

Let's use the small Beatles graph and then move to the slightly larger music dataset that we created from DBPedia (https://docs.stardog.com/tutorials/learn-sparql):



The main query form in SPARQL is a **SELECT** query which, by design, looks a bit like a SQL query. A **SELECT** query has two main components: a list of selected variables and a **WHERE** clause for specifying the graph patterns to match:

```
SELECT <variables>
WHERE {
    <graph-pattern>
}
```

The statements within the WHERE section of SPARQL queries follow the same structure:

```
{ resource propertyName propertyValue . }
```

WHERE section tells the processor which property values to fill into the variables you are selecting and how to pull the data out. SPARQL seeks out statements in the data that match the pattern defined in the WHERE section and binds the data in those statements to the variables.

The basic building block for SPARQL queries is **triple patterns**. A triple pattern is just like an RDF graph triple, but you can use a variable in any one of the three positions. We use triple patterns to find the matching triples in a graph and variables act like wildcards that match any node.

```
SELECT ?album
WHERE {
    ?album rdf:type :Album .
}
```

Thus, **SELECT** query with a single triple pattern: **?album rdf:type :Album**.

This triple pattern will match all the triples in the graph that have **rdf:type** as the predicate and **:Album** as the object. There are three matching triples in our graph so the query result will look like this:

album
:Please_Please_Me
:McCartney
:Imagine

Syntactic simplifications in SPARQL:

- ✓ **WHERE** keyword is optional for **SELECT** queries and can be omitted
- ✓ if all the variables are being selected, then * can be used instead of enumerating them explicitly
- ✓ keyword **a** can be used instead of **rdf:type** (like in RDF)
- ✓ we can omit the trailing **.** for the last triple pattern

With these simplifications our query will be:

```
SELECT * { ?album a :Album }
```

When one or more triple patterns are used together, they form what is known as a Basic Graph Pattern. If you have multiple patterns to match, there is an implicit “AND” statement:

```
SELECT *
{
    ?album a :Album .
    ?album :artist ?artist .
}
```

The second triple pattern in this query will match the triples with **:artist** predicate and we will get a result table with two columns:

album	artist
:Please_Please_Me	:The_Beatles
:McCartney	:Paul_McCartney
:Imagine	:John_Lennon

If you don't want to miss results that match with the first pattern but not with the second one, you could use the **OPTIONAL** keyword for the second pattern. Imagine you want the songs even if there is no length value defined in the database:

```
SELECT ?song ?length
WHERE {
    ?song a :Song .
    OPTIONAL{
         ?song :length ?length .
   }
}
```

If the patterns share the same subject, we can use ; to separate them:

```
SELECT *
{
    ?album a :Album ;
             :artist ?artist ;
             :date ?date .
}
```

album	artist	date
:A_Date_with_Elvis	:Elvis_Presley	"1959-07-24"^^xsd:date
:A_Momentary_Lapse_of_Reason	:Pink_Floyd	"1987-09-07"^^xsd:date
:Achtung_Baby	:U2	"1991-11-18"^^xsd:date
…	…	…

If we want to sort the result, we can add an **ORDER BY**:

```
SELECT *
{
    ?album a :Album ;
             :artist ?artist ;
             :date ?date .
}
ORDER BY ?date
```

Now albums will be returned ordered by their release dates:

album	artist	date
:Elvis_Presley_(album)	:Elvis_Presley	“1956-03-23”^^xsd:date
:Elvis_(1956_album)	:Elvis_Presley	“1956-10-19”^^xsd:date
:Loving_You_(album)	:Elvis_Presley	“1957-07-01”^^xsd:date
…	…	…

If the result of the query is too large, we can limit the results by using the **LIMIT** keyword (only 2 results!). Or we can skip the first N results by adding an **OFFSET N** clause at the end of the query where N is a positive integer.

```
SELECT *
{
    ?album a :Album ;
             :artist ?artist ;
             :date ?date
}
ORDER BY desc(?date)
LIMIT 2
```

album	artist	date
:Hardwired…_to_Self-Destruct	:Metallica	“2016-11-18”^^xsd:date
:24K_Magic_(album)	:Bruno_Mars	“2016-11-18”^^xsd:date

The results can be filtered by using a **FILTER** expression. SPARQL supports many built-in functions for writing such expressions:

- ✓ comparison operators: (=, !=, <, <=, >, >=)
- ✓ logical operators (&&, ||, !)
- ✓ mathematical operators (+, -, /, *)

If we want only the albums released after 1970 (included):

```
SELECT *
{
    ?album a :Album ;
             :artist ?artist ;
             :date ?date
    FILTER (?date >= "1970-01-01"^^xsd:date)
}
ORDER BY ?date
```

album	artist	date
:This_Girl\'s_in_Love_with_You	:Aretha_Franklin	“1970-01-15”^^xsd:date
:Chicago_(album)	:Chicago_(band)	“1970-01-26”^^xsd:date
:Morrison_Hotel	:The_Doors	“1970-02-09”^^xsd:date
…	…	…

If we want an extra variable that is not included into the triple patterns (an extra column in the result):

```
SELECT ?album ?artist ?newVariable
WHERE {
    ?album rdf:type :Album .
    ?album rdf:artist ?artist .
    BIND(“some value” AS ?newVariable)
}
```

album	artist	newVariable
:A_Date_with_Elvis	:Elvis_Presley	some value
:A_Momentary_Lapse_of_Reason	:Pink_Floyd	some value
:Achtung_Baby	:U2	some value
…	…	…

If we need to remove duplicates from the results, **DISTINCT** keyword should be used after **SELECT**.

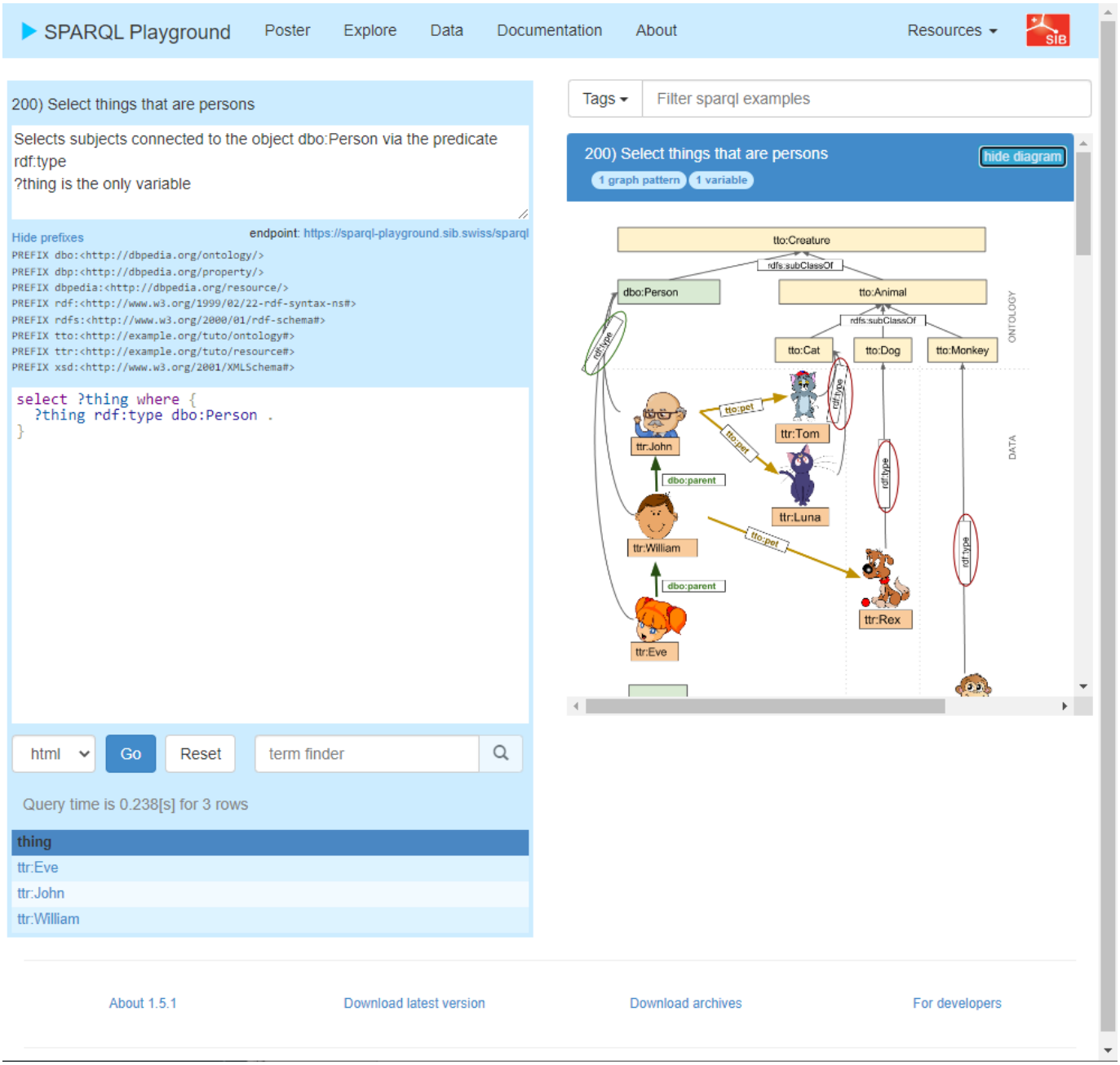
SPARQL Playground – Web interface to SPARQL

Let's see some examples with results using a free online tool for SPARQL: https://sparql-playground.sib.swiss/ (now it is not available but the examples are useful).

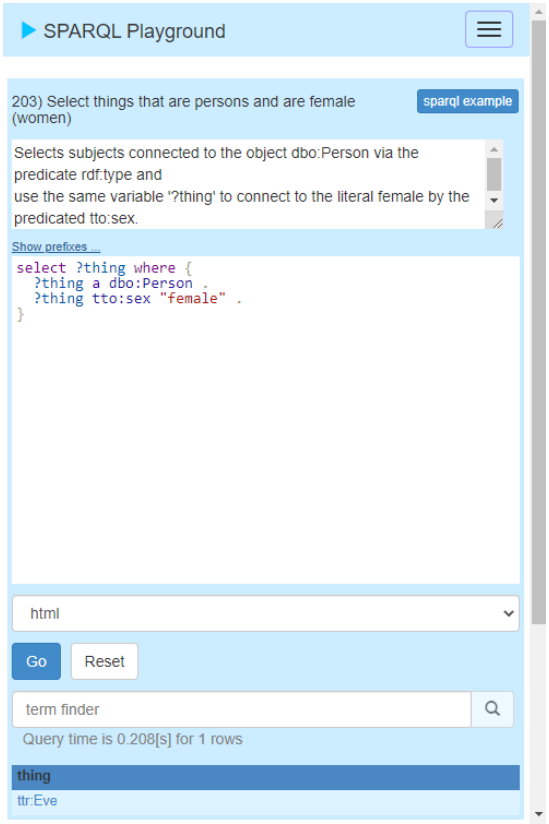
- The default **endpoint** is https://sparql-playground.sib.swiss/.
- The prefixes are listed above.

This SPARQL query is asking for all the persons in this database.

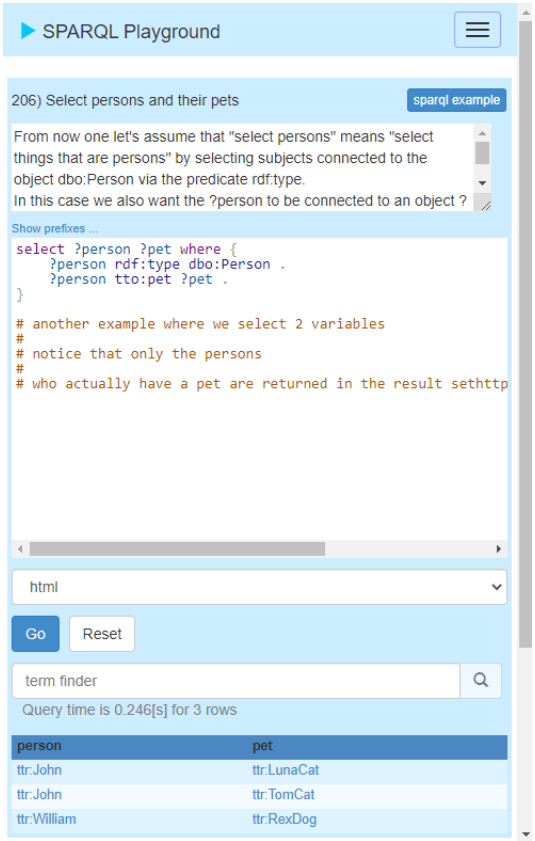
- Pressing Go button you can obtain the result.
- In the right part of the window, you have a list of examples where you can obtain the graph definition using show diagram button.



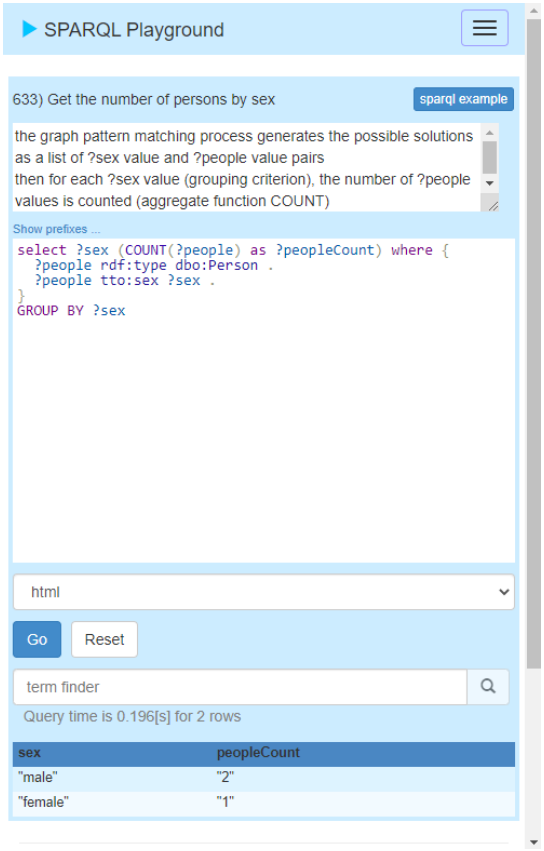
If you want the only persons that are women/female:



Select persons and their pets:



Get the number of people by sex (grouping data results):



If we want to make a union of two information, we can use UNION or specific VALUES:

https://dbpedia.org/sparql/

Searching on language-tagged literal values that are the objects of **rdfs:label** properties: select athlete with the name “Cristiano Ronaldo”:

```
SELECT * WHERE {
 ?athlete rdfs:label     "Cristiano Ronaldo"@en
}
```

SPARQL | HTML5 table

athlete
http://dbpedia.org/resource/Category:Cristiano_Ronaldo
http://dbpedia.org/resource/Cristiano_Ronaldo

* Use https://dbpedia.org/page/Cristiano_Ronaldo to add new properties to the query!

TO DO

You will execute several SPARQL queries. Please check the Tasks.