PhUSE CSS
Linked Data and Graph Database
Hands-on Workshop
EXERCISES

Version 1.0

March 2017

Contents

Introduction and Disclaimer

Instructions in this document are specific to the cloud server instance used for the PhUSE CSS workshop. The exercises represent one of many possible approaches to the material and make no claim to be best or recommended method.

Your feedback is welcomed and encouraged. Please send your comments to:

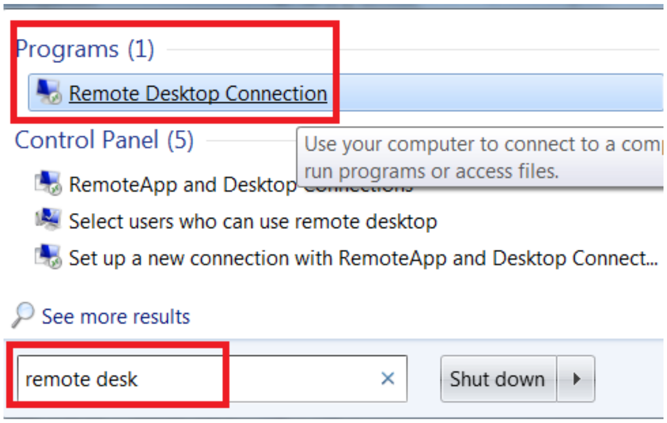
Tim Williams
tim.williams@PhUSE.eu

Server Login and Preparation

Login to the cloud server provided for the exercises. Instructions assume Windows OS on your local machine.

Connect to the server using Remote Desktop

1. In the search box on the taskbar, type **remote desk,** then select **Remote Desktop Connection** from the Programs section.


Programs (1)
Remote Desktop Connection
Control Panel (5)
Use your computer to connect to a comp
run programs or access files.
RemoteApp and Desktop Connections
Select users who can use remote desktop
Set up a new connection with RemoteApp and Desktop Connect...
See more results
Shut down

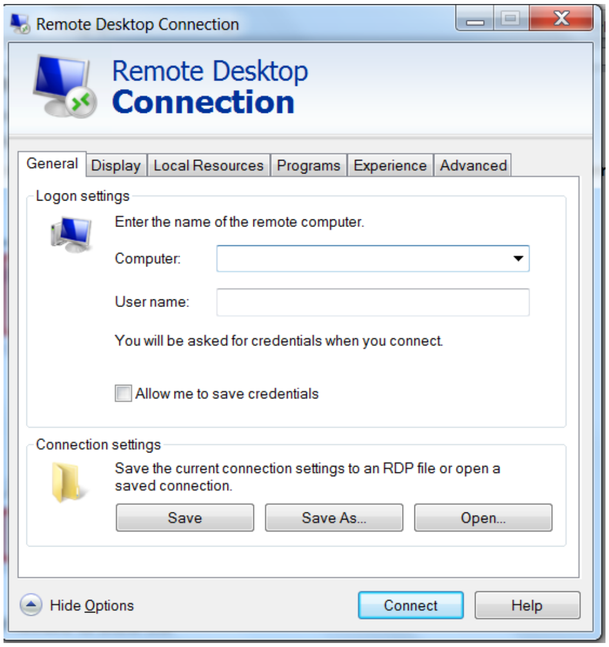
remote desk

2. Click **Show Options** if needed to show the fields for **Computer:** and **User name:**
   You will be provided with an IP Address for the Computer: field. Everyone will use the same User name:

   Computer: ________________
   (Provided by Instructor)

   User name: `phusecss`

   Password: __________________
   (Provided by Instructor)



Enter the name of the remote computer.

You will be asked for credentials when you connect.

Save the current connection settings to an RDP file or open a saved connection.

3. Click **Connect** after entering the Computer IP address and your username for the session.
4. Enter the password supplied by the instructor and click **OK**.

STOP	Stop here and wait for the instructor. Presentation follows	STOP

Exercises

The exercises are presented in three sections.

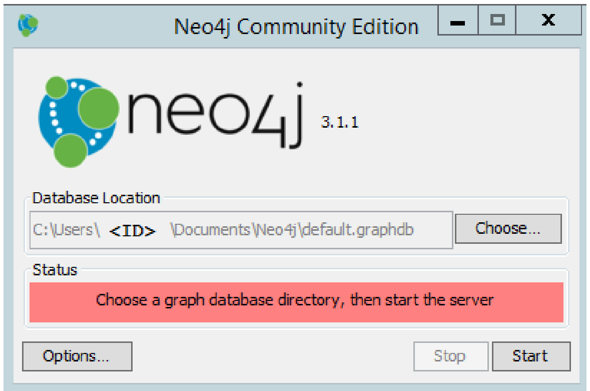
Section 1: Introduction to Neo4j using an abbreviated graph of the Simpsons family.
Section 2: Introduction to Resource Description Framework (RDF) using the same Simpsons family data.
Section 3: Federated Query (time permitting or take-home exercise).

1. Simpsons Family in Neo4j

1.1 Explore

1. Start Neo4j by clicking on the application icon.

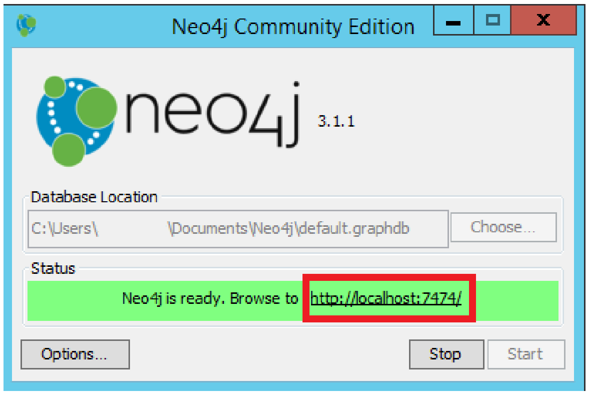
2. Accept the default **Database Location** shown in the dialog box and click **Start**.



Database Location: C:\Users\ <ID> \Documents\Neo4j\default.graphdb

Status: Choose a graph database directory, then start the server

3. There will be a delay while the database initiates, then the red bar changes to green and contains the address of the Neo4j instance. Click on the **Browse to** URL to launch Neo4j in a web browser.



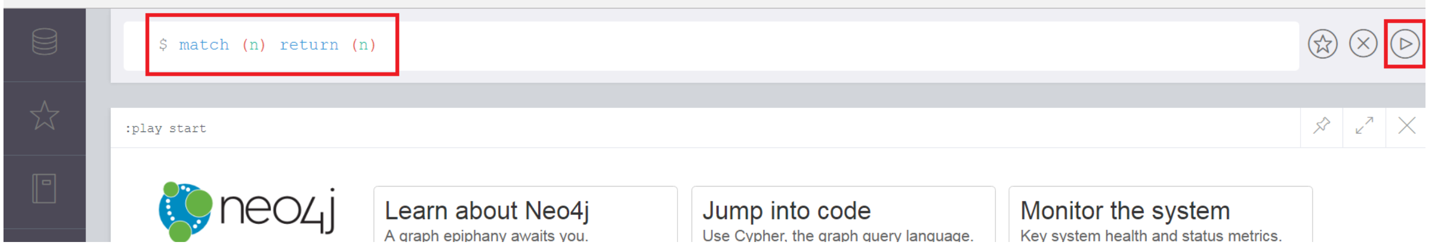
Database Location: C:\Users\ \Documents\Neo4j\default.graphdb

Status: Neo4j is ready. Browse to http://localhost:7474/

The graph contains an incomplete representation of the Simpsons Family.

4. Execute the following query by typing it into the query editor window of Neo4j and clicking the ▷ icon:

```
match (n) return (n)
```



```
$ match (n) return (n)
```

```
:play start
```

neo4j
Learn about Neo4j
A graph epiphany awaits you.
Jump into code
Use Cypher, the graph query language.
Monitor the system
Key system health and status metrics.

5. Explore the graph:
   a. Drag the nodes to arrange them.
   b. Click on nodes to select them and see their properties at the bottom of the window.
   c. Observe the relations between Marge and her children.
   d. Explore the relation between Marge and Maggie. **Who was the doctor that delivered Maggie?**

1.2 Query

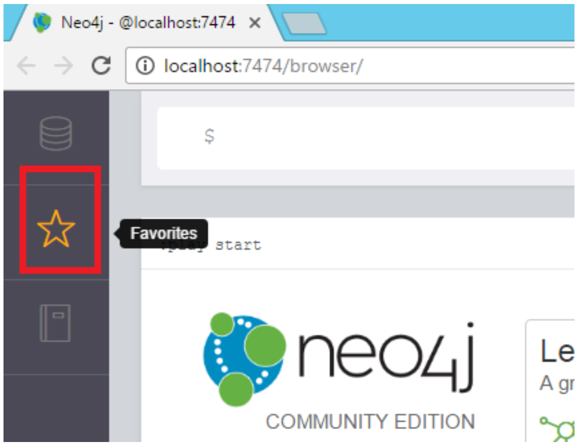
There are multiple ways to execute a query in Neo4j, including (but not limited to):

- Write the query directly in query editor window.
- Execute a "Saved Script".
- Drag a .cql file into the import section of the "Saved Scripts" sidebar, then:
  - Execute from the sidebar.
  - Execute from the query editor window after selecting the script from the sidebar.

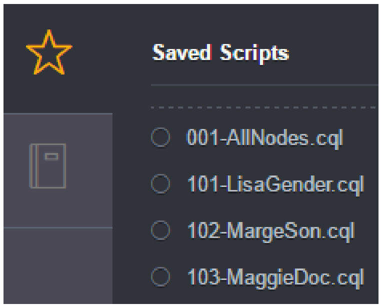
The steps in these exercises use Saved Scripts previously prepared by the instructor. Experiment with the other methods as time allows. You can revisit the graph anytime by executing the following query (also available as a Saved Script).

```
match (n) return (n)
```

1. Click on the star in the application side bar to view the **Saved Scripts** prepared for these exercises.



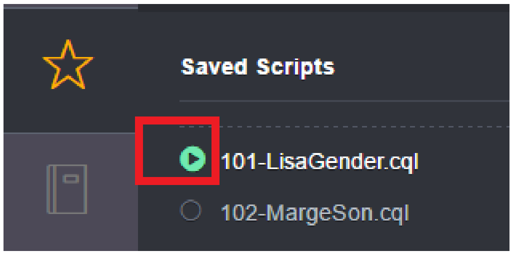
   Please ask for assistance if you do not see the **Saved Scripts**.



Saved Scripts can be executed in different ways:

Method 1: From the sidebar

a) Click on the execute button beside the name of the script and the query and result will appear in the application.

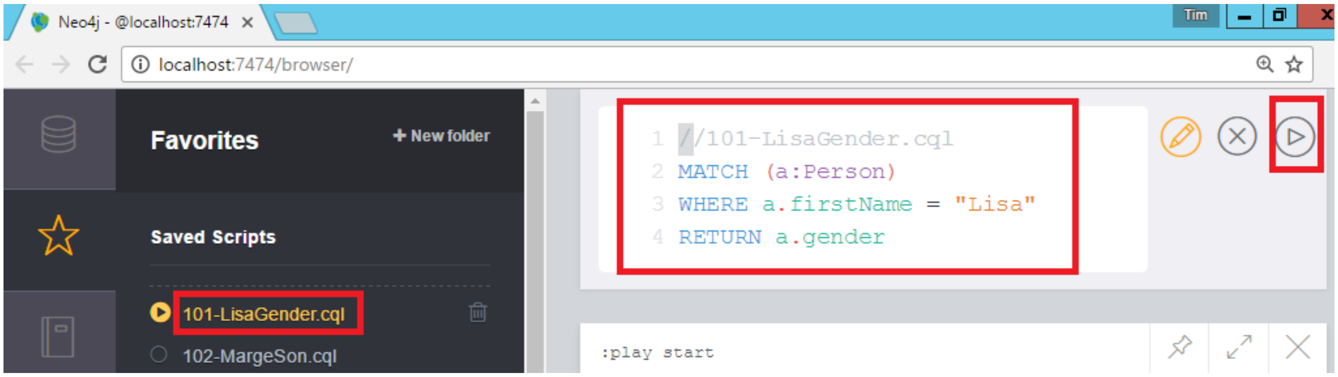


<u>Method 2: From the Execution Window</u>

a. Click on the **<u>name</u>** of the Saved Script. This places the cypher code in the execution window.

b. Click on the execution button to the right of the query text to execute the query.

Method 2 is preferred for these exercises because it allows viewing of the query prior to execution.



```
//101-LisaGender.cql
MATCH (a:Person)
WHERE a.firstName = "Lisa"
RETURN a.gender
```

2. Follow along with the instructor to execute the following queries.

- **Query a node property:** What Gender is Lisa?
  Note how nodes were assigned a "Person" type when they were created so you can differentiate them from other types of *things* that may be represented in the graph.

101-LisaGender.cql

```
MATCH (a:Person)
WHERE a.firstName = "Lisa"
RETURN a.gender
```

- **Query a Relation:** Who is Marge's Son?

102-MargeSon.cql

```
MATCH (pers1)-[:hasSon]-(pers2)
WHERE pers1.firstName='Marge'
RETURN pers2.firstName
```

- **Query a relation property:** Who delivered Maggie?

103-MaggieDoc.cql

```
MATCH (pers1)-[r:hasDaughter]-(pers2)
WHERE pers2.firstName='Maggie'
RETURN r.DeliveredBy as DeliveryDoctor
```

Stop here and wait for the instructor.

Presentation follows

1.3 Create

In this section you will create a node for Homer Simpson and the relationships to his children.

1. Create a "Person" node for Homer and the nodes properties : firstName, lastName, Gender.

   104-CreateHomer.cql

```
CREATE (:Person{
           firstName:"Homer",
           lastName:"Simpson",
           gender: "Male"})
```

2. Create the relations between Homer and his children. Note the directionality in the relation.

   105-CreateRel-HomerDaughter.cql

```
MATCH (person1), (person2)
WHERE person1.firstName="Homer" AND
    (person2.firstName="Lisa" OR
     person2.firstName="Maggie")
CREATE (person1)-[:hasDaughter]->(person2);
```

   106-HomerToSon.cql

```
MATCH (person1), (person2)
WHERE person1.firstName="Homer" AND
     person2.firstName="Bart"
CREATE (person1)-[:hasSon]->(person2);
```

3. Explore the revised graph.

```
    match (n) return (n)
```

- Arrange the graph to explore the new node and relations.

1.4 Query

1. **Who are Homer's children (hasSon or hasDaughter)?**

   There is no "hasChild" relation in the data, so both hasSon and hasDaughter must be specified in the query.

   107-HomerChildren.cql

```
MATCH (person1)-[r]->(person2)
WHERE (person1.firstName ='Homer' AND
       (type(r) = 'hasSon' OR
        type(r) = 'hasDaughter'))
RETURN DISTINCT person2.firstName
```

2. **How many children does Homer have??**

   a**. Use the keyboard "cursor up" key to return the previous query**.

   b. Edit the RETURN line in the query to read:

```
//…other lines except RETURN from previous…
RETURN COUNT(person2)
```

c) Execute the revised query.

3. **Who are parents?**
   In this data model, parents have at least one *hasSon* or *hasDaughter* relation. Note the direction of the relation and use of DISTINCT.

108-Parent.cql

```
MATCH (person1)-[r]->(person2)
WHERE (type(r) = "hasSon" OR
       type(r) = "hasDaughter")
RETURN DISTINCT person1.firstName
```

4. **Who is Lisa's brother?**

Lisa's brother shares the same parent with Lisa. Lisa is the daughter of this parent; her brother is the son of this same person.

Return the graph pattern.

109-LisaBrother-Graph.cql

```
MATCH a = (brother)-[:hasSon]-(parent)-
[:hasDaughter]-(daughter)
WHERE daughter.firstName='Lisa'
RETURN a
```

Return the name value. Refer back to the graph result to see why DISTINCT is used.

110-LisaBrother-Name.cql

```
MATCH a = (brother)-[:hasSon]-(parent)-
[:hasDaughter]-(daughter)
WHERE daughter.firstName='Lisa'
RETURN DISTINCT brother.firstName
```

5. **Who are Bart’s sisters?**

Follow the same graph pattern and merely change the WHERE and RETURN statements as follows:

Return the graph pattern.

111-BartSister-Graph.cql

```
MATCH a = (brother)-[:hasSon]-(parent)-
[:hasDaughter]-(daughter)
WHERE brother.firstName='Bart'
RETURN a
```

Return the name. How many names would be returned if DISTINCT was omitted?

112-BartSister-Name.cql

```
MATCH a = (brother)-[:hasSon]-(parent)-
[:hasDaughter]-(daughter)
WHERE brother.firstName='Bart'
RETURN DISTINCT daughter.firstName
```

Continue to explore the graph with your own queries while the other attendees catch up.

STOP	Stop here and wait for the instructor. Presentation follows	STOP

This is the end of exercises for Neo4j. RDF will be introduced before returning to the exercises in the next section.

2. Simpsons Family in RDF

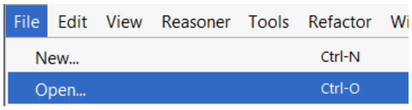
The ontology used in this exercise was designed to illustrate specific concepts. It is largely incomplete and in many cases does not follow "best practices." Consider it a starting point for understanding RDF, ontologies, and the Protégé application.

2.1 Explore

Load the Simpsons family ontology and instance data into Protégé.

1. Start **Protégé** by double-clicking on application shortcut on the desktop . There will be a short delay while the application initializes.

2. Select **File | Open**

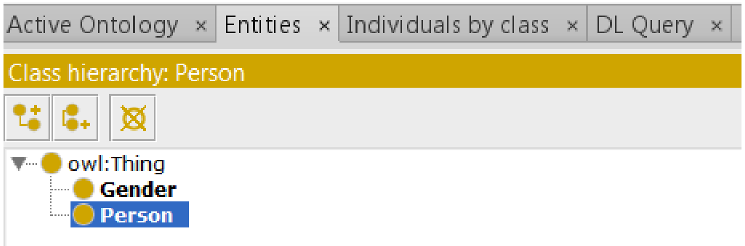


3. Browse to the folder: **C:\PhUSECSS\data**
4. Open the file **simpsons.owl** .

<u>**What is defined in the ontology?**</u>

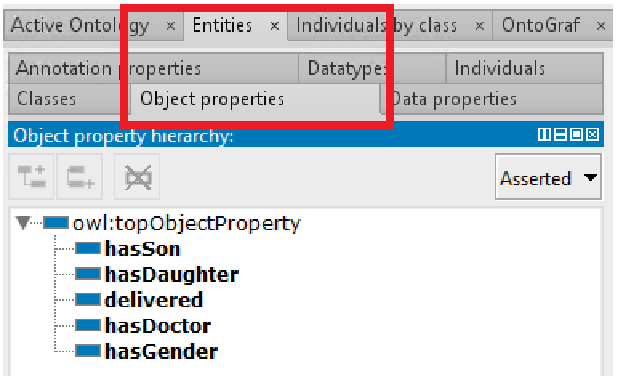
- **Entities**

5. Select the **Entities** tab.
6. In the **Class Hierarchy** window, expand the **owl:Thing** class to view the two subclasses: *Gender*, *Person*. These are the *types* of "Things" defined in the ontology.



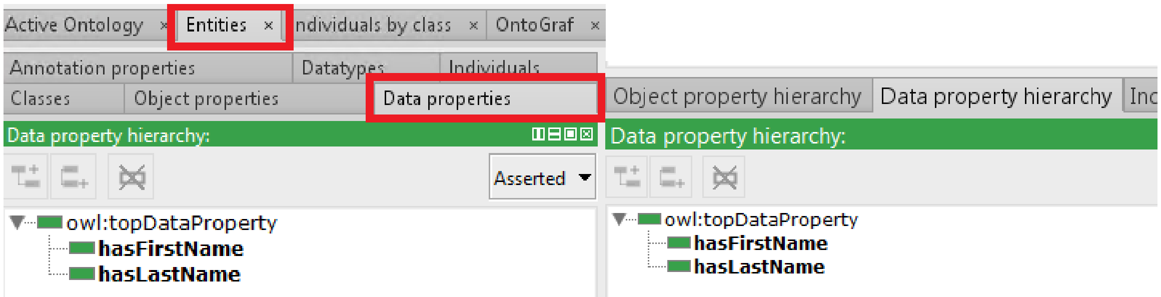
- **Object Properties**

7. Select the **Object properties** tab and expand **owl:topObjectProperty** to see the properties associated with the Objects in our Simpsons knowledgebase.



- **Data Properties**

8. Select the **Data properties** tab and expand **owl:topDataProperty** to see the subproperties. Data properties relate literal data values to objects.

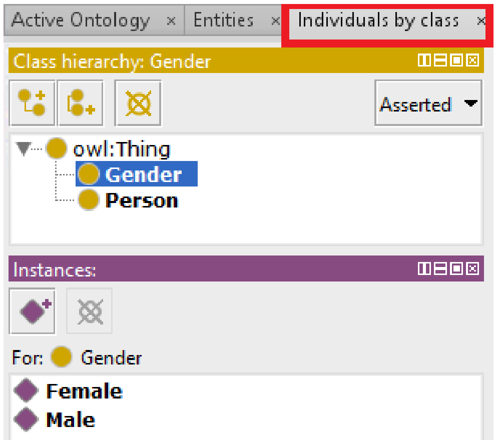


owl:topDataProperty
hasFirstName
hasLastName

- **Instances Data**

9. Select **Individuals by class** and expand **owl:Thing** to see the Gender and Person subclasses.
10. Click on **Gender** to observe two instances of gender in the Instances class:

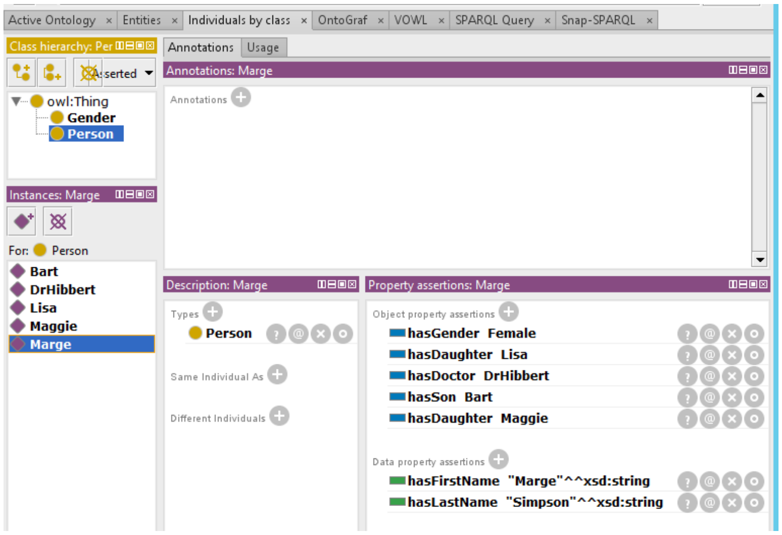
11. Click on the **Person** subclass to display the five instances of people defined in the data.



owl:Thing
Gender
Person

For: Gender
Female
Male

12. Click on **Marge** in the **Instances** window to see how she is connected to the Person class (Marge *is a* Person) and how she relates to other instances in the data with property assertions *hasGender*, *hasDaughter*, etc. and Data property assertions *hasFirstName*, *hasLastName*.



owl:Thing
Gender
Person

For: Person
Bart
DrHibbert
Lisa
Maggie
Marge

Types: Person

Object property assertions
hasGender Female
hasDaughter Lisa
hasDoctor DrHibbert
hasSon Bart
hasDaughter Maggie

Data property assertions
hasFirstName "Marge"^^xsd:string
hasLastName "Simpson"^^xsd:string

13. Explore the other Person instances to find the answers to these questions:
    a. Who is the doctor that delivered Maggie? (Find the answer without using a query. We will later answer this question using SPARQL.)
    b. How does this model differ from the Labeled Property Graph model of representing the same fact?

Ontology Visualization

1. VOWL
    a. Click on the *VOWL* tab to visualize the components of the ontology.

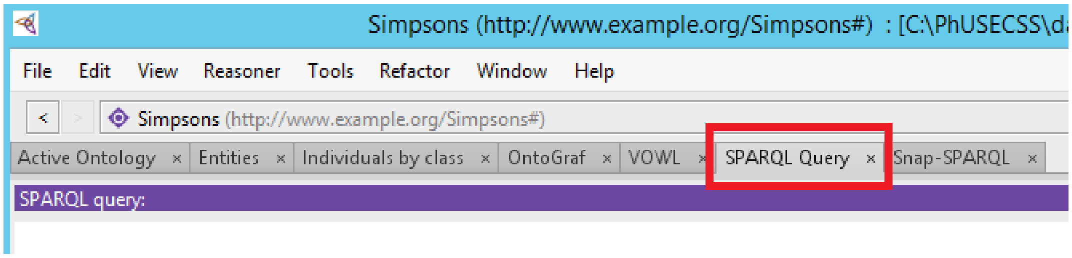
   b. Explore the relationships between the different entities. Are the relationships correct? What would you add to increase the usefulness and validity of this ontology?

2. OntoGraf
   a. Click on the Ontograf tab to visualize the ontology and instances.
   b. Expand node relations by double-clicking nodes that have + signs.
   c. Mouse-over the lines (edges) to view the relationships.

2.2 Query

The exercises in this section require copy-paste from .rq files into the SPARQL Query window in Protégé.

1. Locate the SPARQL scripts in the folder: **C:\PhUSECSS\scripts**
2. Double click on the first SPARQL script **200-ShowTriples.rq** to open it into Notepad++.
3. Copy the file contents to the clipboard (***ctrl-a ctrl-c***) .
4. Click on the **SPARQL Query** tab (not the Snap-SPARQL tab! We will us that one later).
   Ask for assistance if the SPARQL tab is missing. Do NOT use the Snap SPARQL tab!



5. Paste the clipboard content into the SPARQL Query window (***ctrl-v***) .
6. Click **Execute** to perform the query.
7. Repeat these Open-Copy-Paste-Execute steps for each query in the following exercises.

Follow along with the instructor as they explain and execute each query.

- **Show the triples**
  *Compare this approach with the **`match (n) return (n)`** of Cypher.*

200-ShowTriples.rq

```
PREFIX simpsons: <http://www.example.org/Simpsons#>
SELECT *
WHERE {
    ?s ?p ?o
}LIMIT 10
```

If you receive and error, ensure you are using the SPARQL-Query tab and not Snap-SPARQL!

- **Query a relation:** What Gender is Lisa?
  Compare with the node property query of Cypher.

201-LisaGender.rq

```
PREFIX simpsons: <http://www.example.org/Simpsons#>
SELECT ?gender
WHERE {
    simpsons:Lisa simpsons:hasGender ?gender
}
```

- **Query a relation**: Who is Marge's Son?
  Compare with the relation query of Cypher.

202-MargeSon.rq

```
PREFIX simpsons: <http://www.example.org/Simpsons#>
SELECT ?sonFirstName
WHERE {
    ?person simpsons:hasSon ?son ;
            simpsons:hasFirstName ?parentFirstName .
    ?son    simpsons:hasFirstName ?sonFirstName
    FILTER(regex(?parentFirstName,  "^Marge"))
}
```

Note: The query was written in this way to illustrate graph traversal and FILTER instead of using simpsons:Marge as the starting subject.

- **Query a relation:** Who delivered Maggie?
  - STR() used to clean up the strings
  - "Dr." added to result using BIND and does not became a part of the graph data.

203-MaggieDoc.rq

```
# NB: Protege uses SPARQL1.0 so CONCAT() not avail.
PREFIX simpsons: <http://www.example.org/Simpsons#>
SELECT ?title ?docFirstName  ?docLastName
WHERE {
    ?doc simpsons:delivered simpsons:Maggie ;
             simpsons:hasFirstName ?docFirstName_ ;
             simpsons:hasLastName ?docLastName_ .
  BIND( STR(?docFirstName_) AS ?docFirstName  )
  BIND( STR(?docLastName_) AS ?docLastName  )
  BIND("Dr.  " AS ?title)
}
```

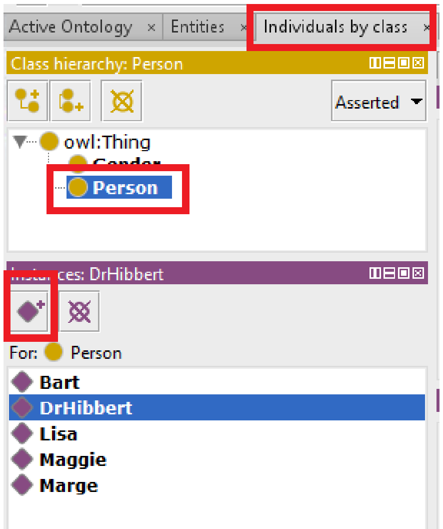
Stop here and wait for the instructor.

Presentation follows

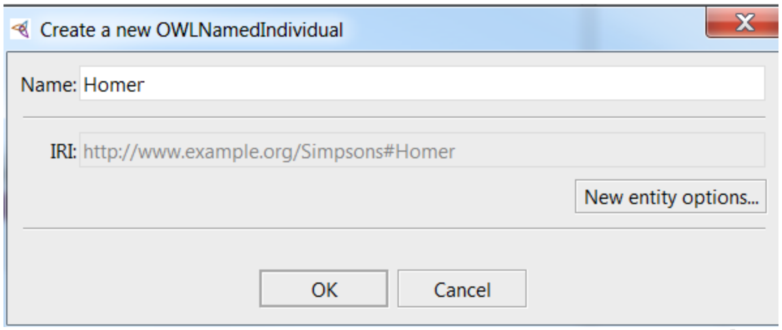
2.3 Create

In this section you will create a node for Homer Simpson and the relationships to his children.

1. Create a node for Homer and the node properties.
   a. Select the **Individuals by Class** tab and click on **Person** in the **Class hierarchy** window.



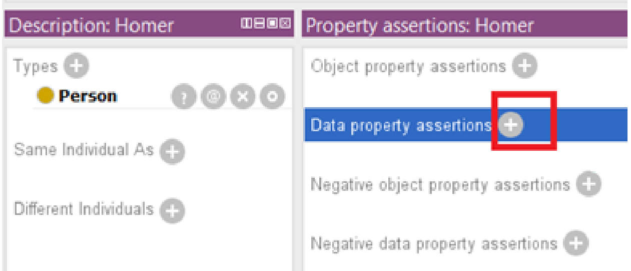
   b. In the **Instances** window, click on the icon to add a new individual.
   c. Enter Homer in the **Name:** field and click **OK.**



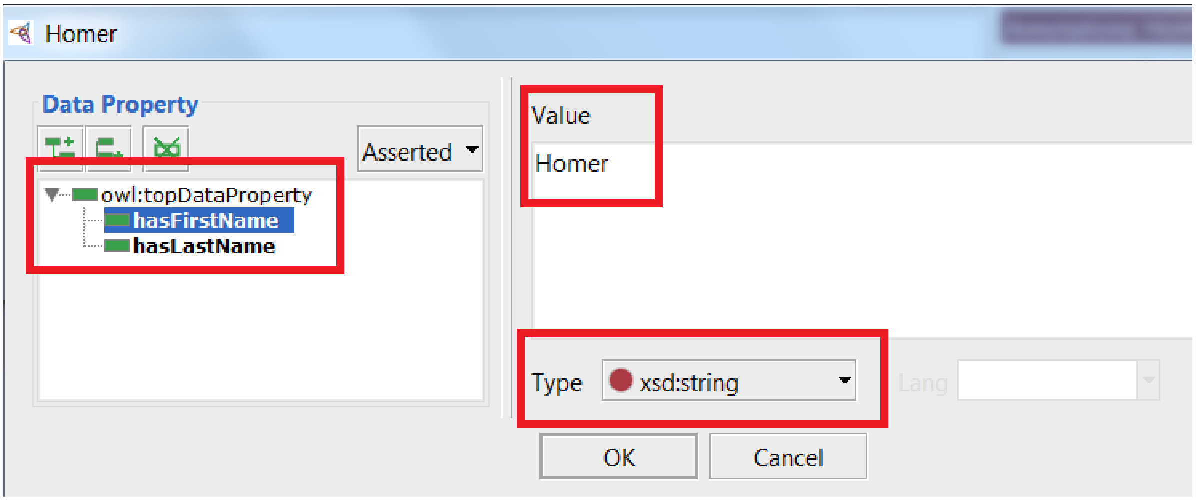
Name: Homer

IRI: http://www.example.org/Simpsons#Homer

2. Add ***Data Property Assertions*** for Homer: firstName and lastName.
   a. In the window **Property assertions: Homer**, click the ⊕ beside the **Data property assertions**.


Description: Homer
Types
Person
Same Individual As
Different Individuals
Property assertions: Homer
Object property assertions
Data property assertions
Negative object property assertions
Negative data property assertions

   b. In the resulting dialog box, expand **owl:topDataProperty** to show the properties *hasFirstName* and *hasLastName*.
   c. Select ***hasFirstName*.**
   d. In the Value field, type the text: Homer
   e. In the **Type** selection box, choose **xsd:string** .
   f. Click **OK**.
   g. Repeat steps *a-f* to add the property ***hasLastName*** with a value of Simpson as **xsd:string**

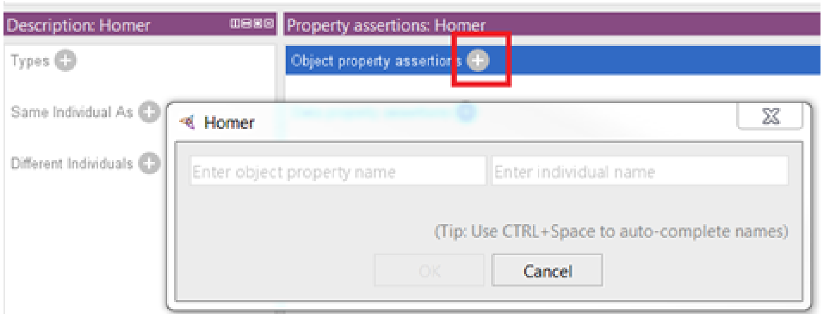


Value: Homer

Type: xsd:string

3. Add ***Object Property Assertions*** for Homer's relationships to **Children** and **Gender**.

   a. In the Property assertions window for Homer, click the + beside **Object property assertions** to obtain the popup box.

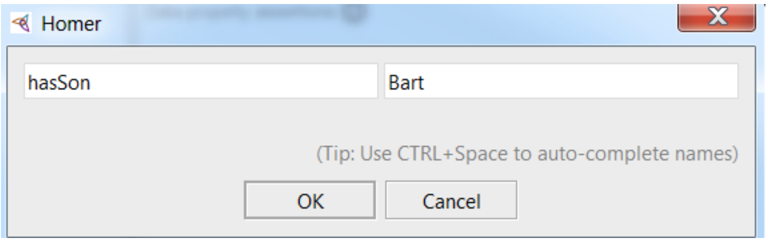

Description: Homer
Types
Same Individual As
Different Individuals
Property assertions: Homer
Object property assertions
Homer
Enter object property name
Enter individual name
(Tip: Use CTRL+Space to auto-complete names)
OK
Cancel

In the left field enter: ***hasSon***

   b. In the right field enter: **Bart**

   c. Then click **OK.**

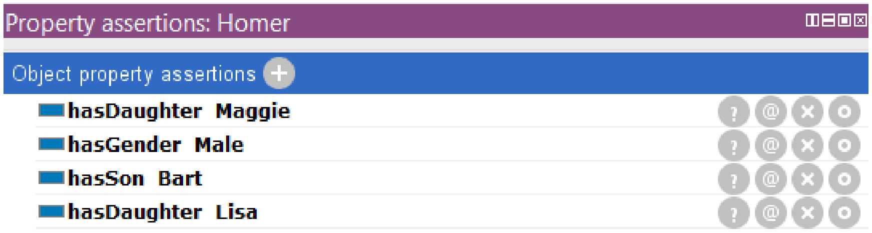
   CAUTION: Entries are case sensitive.



hasSon

Bart

   d. Repeat for adding ***hasDaughter*** **Lisa**, ***hasDaughter*** **Maggie**, and ***hasGender*** **Male**.



hasDaughter Maggie

hasGender Male

hasSon Bart

hasDaughter Lisa

2.4 Query

Remember to copy-paste into the **SPARQL Query** window for these steps.

1. **Who are Homer's children (hasSon or hasDaughter)?**

   Query uses hasSon, hasDaughter for comparison with Neo4j:

204-HomerChildren.rq

```
PREFIX simpsons: <http://www.example.org/Simpsons#>
SELECT ?child
WHERE
{
    {simpsons:Homer simpsons:hasDaughter ?child . }
    UNION
    {
        {simpsons:Homer simpsons:hasSon ?child . }
    }
}
```

2. **How many children does Homer have?**
   The SELECT statement now includes the COUNT function.

205-HomerChildCount.rq

```
PREFIX simpsons: <http://www.example.org/Simpsons#>
SELECT  (COUNT(?child) AS ?count)
WHERE
{
    {simpsons:Homer simpsons:hasDaughter ?child . }
    UNION
    {
        {simpsons:Homer simpsons:hasSon ?child . }
    }
}
```

3. **Who are parents?**
   Find occurrences of both hasSon and hasDaughter in case a parent only has one of these types of relations. A later solution will employ reasoning. UNION acts as a 'OR' match.

206-Parent.rq

```
PREFIX simpsons: <http://www.example.org/Simpsons#>
SELECT  DISTINCT ?parent
WHERE
{
    {?parent simpsons:hasDaughter ?child . }
    UNION
    {?parent simpsons:hasSon ?child . }
}
```

4. **Who is Lisa's brother?**

207-LisaBrother.rq

```
PREFIX simpsons: <http://www.example.org/Simpsons#>
SELECT  DISTINCT ?brother
WHERE
{
    ?parent simpsons:hasDaughter simpsons:Lisa ;
                  simpsons:hasSon  ?brother .
}
```

Stop here and wait for the instructor.

Presentation follows

2.5 Extend with OWL

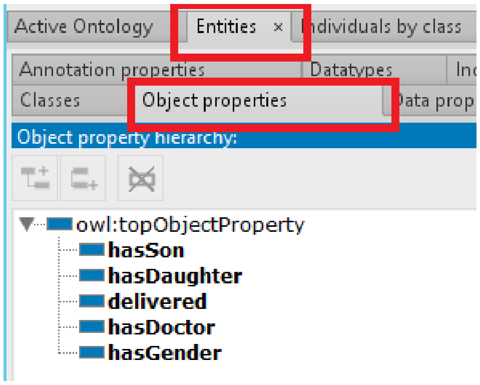
In this section you will add logic on top of the existing data that allows you to query and interpret it in new ways using the new and inferred relations.

Many popular triplestores allow SPARQL queries to leverage the logic encoded in OWL ontologies. **In the following exercises you must use the Snap SPARQL plugin tab,** NOT the SPARQL Query tab used in the previous section.

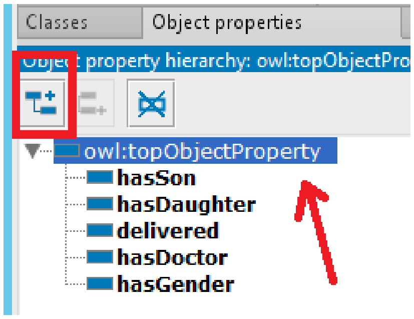
2.5.1 Create new Object Properties

Create *hasChild* as a Parent Property for *hasDaughter, hasSon*

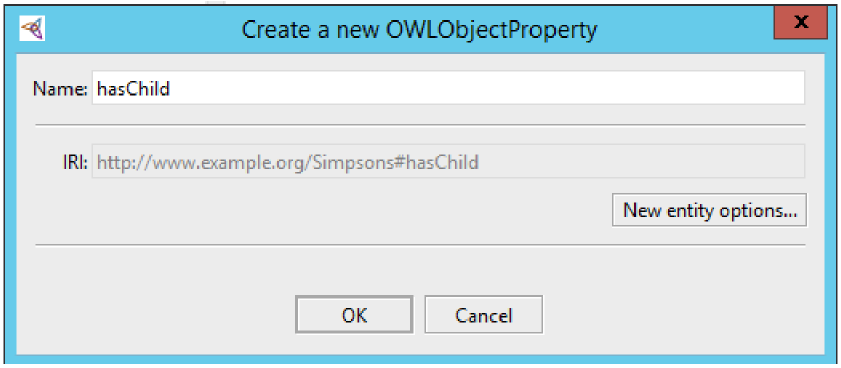
1. Click on the **Entities** tab, then the **Object Properties** tab.



2. Select **owl:topObjectProperty** (indicated with red arrow) and then click the create subproperty icon (indicated with red square.



3. In the resulting dialog box, type: **hasChild** . Note the capital C in for the naming convention. Click **OK**.

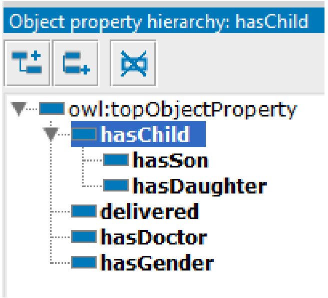

Create a new OWLObjectProperty

Name: hasChild

IRI: http://www.example.org/Simpsons#hasChild

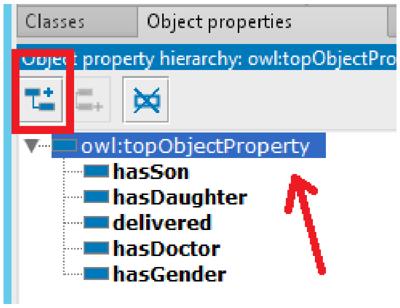
4. Drag and drop both **hasDaughter** and **hasSon** properties under **hasChild** so they appear as:

   hasDaughter and **hasSon** are both subproperties of **hasChild** ; both are a "type of" **hasChild** relationship.

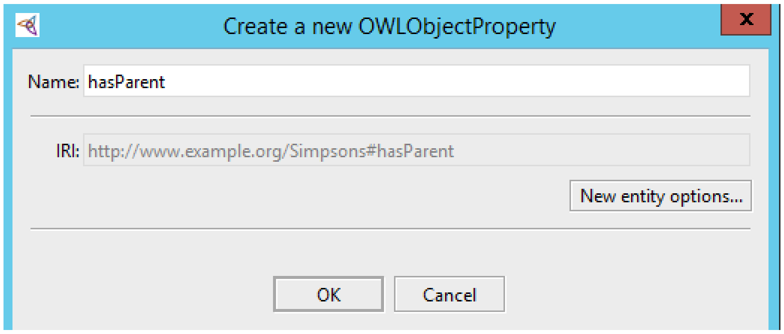


Create *hasParent* as Inverse of *hasChild*

1. Select **owl:topObjectProperty** (indicated with red arrow) and then click the create subproperty icon (indicated with red square.



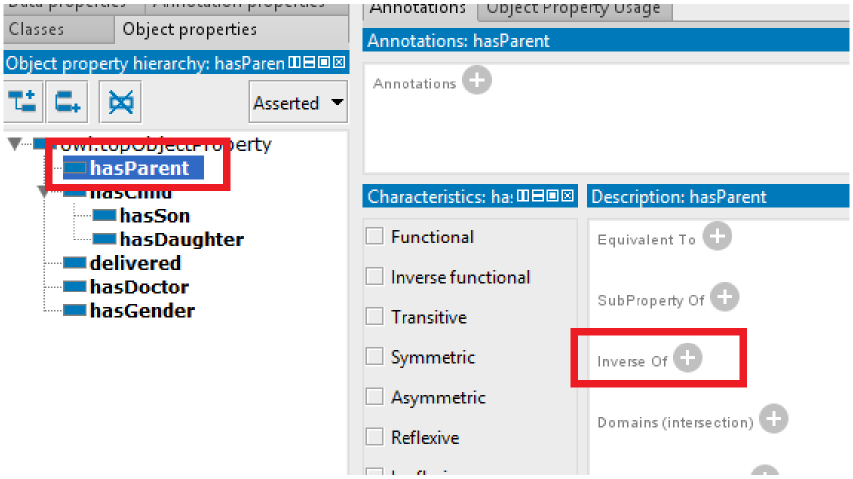
2. In the resulting dialog box, type: **hasParent** . Note the capital P in for the naming convention. Click **OK**.



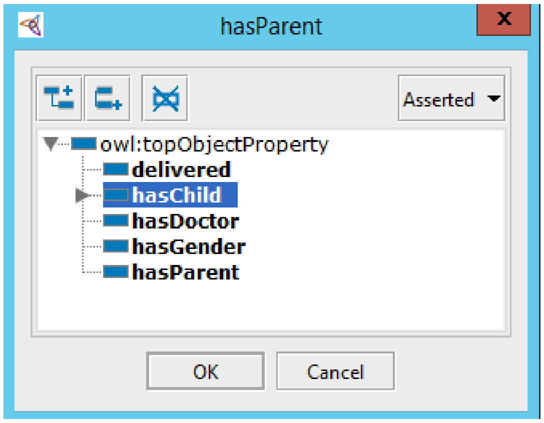
Name: hasParent

IRI: http://www.example.org/Simpsons#hasParent

3. Select the new **hasParent** property, then Click ⊕ next to **Inverse of**.



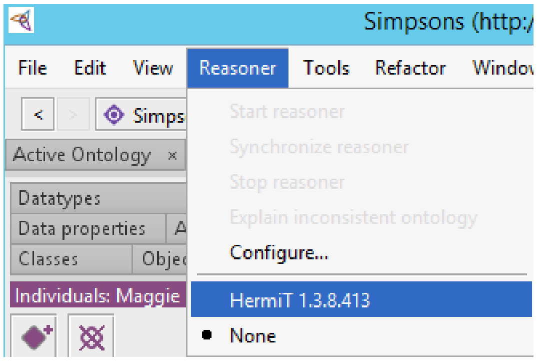
4. In the resulting dialog box, choose **hasChild**, then click **OK**.



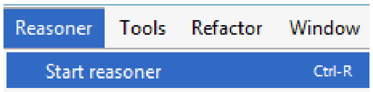
Now the **hasChild** and **hasParent** relations can be used in query engines that support Inferencing.

2.5.2 Apply a Reasoner

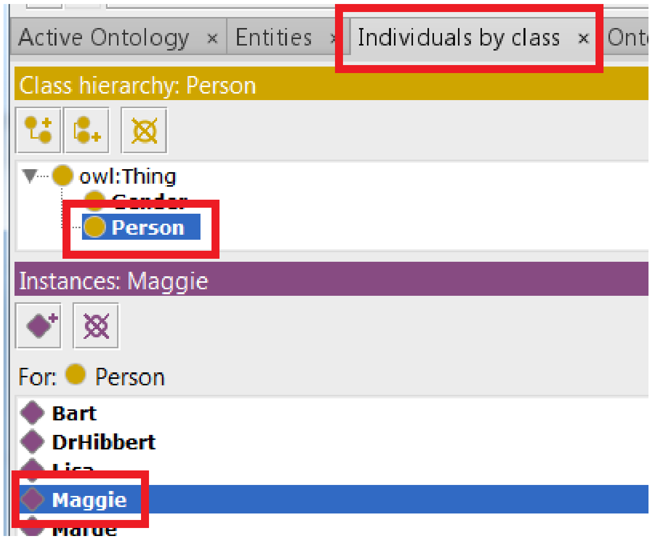
1. From Protégé's the top menu select **Reasoner** and then choose **HermiT**.



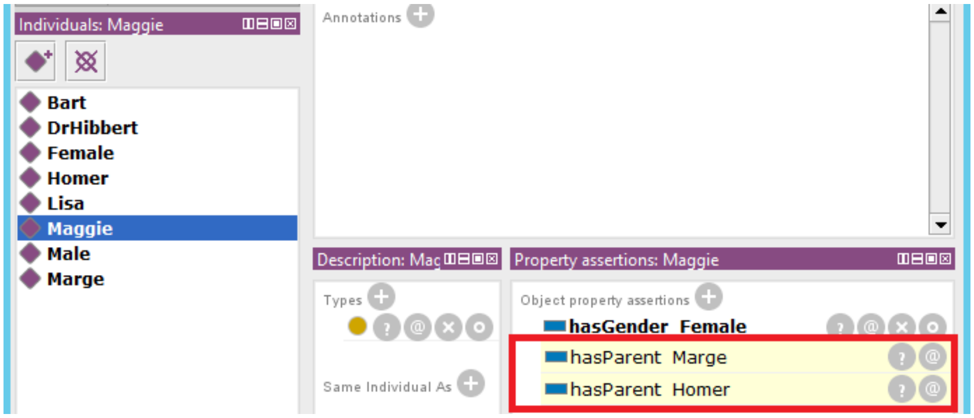
2. Select the **Reasoner** menu again and choose **Start reasoner**.



3. Select the menu tab **Individuals by Class.** Select **Person** in the **Class hierarchy** window, then **Maggie** in the **Instances** window:



4. Observe how relationships are now inferred in the data. The original data contained no relationship FROM Maggie to Marge, only a Marge to Maggie relation using ***hasDaughter***. The data can now be explored in new ways!



hasGender Female
hasParent Marge
hasParent Homer

2.5.3 Query Inferred Relations

Reminder: use Snap-SPARQL window!

1. **Determine who are parents using hasChild, the parent property of hasDaughter, hasSon.**

   208-ParentAsHasChild-Reas.rq

```
PREFIX simpsons: <http://www.example.org/Simpsons#>
SELECT DISTINCT ?parent
WHERE {
    ?parent simpsons:hasChild ?child
}
```

2. **Determine who are parents using hasParent, the inverse of the hasChild property.**

   209-ChildhasParent-Reas.rq

```
PREFIX simpsons: <http://www.example.org/Simpsons#>
SELECT DISTINCT ?parent
WHERE {
    ?child simpsons:hasParent ?parent
}
```

For those who finish early: Try adding new relations and rules while the rest of the class catches up.

STOP	Stop here and wait for the instructor. Presentation follows	STOP

3. Federated Query

1. Open Google Chrome and click on the **Finki SPARQL** bookmark or navigate to the address:

   http://linkeddata.finki.ukim.mk/sparql

2. Copy-paste the file C:\PhUSECSS\scripts\300-RelatedDrugs.rq into the query window opened step 1.
3. Execute the query. Wait patiently…
4. View the results of the query that federated data from the Finki, Mannheim DrugBank and DbPedia endpoints for the drug Duloxetine.
5. Scroll to the right to see results that include Food and Drug interactions and other information not available in all of the datasets.

Other queries of interest (most not federated) are available at: http://seminant.com/queries

---- END OF EXERCISES ----