Ontology-based Data Access: From Practice to Theory

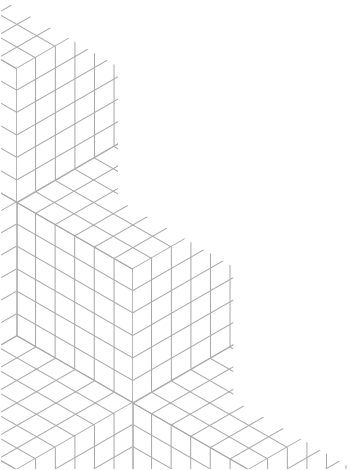
Diego Calvanese, Martin Rezk, Guohui Xiao
Faculty of Computer Science
Free University of Bozen-Bolzano, Italy

Freie Universität Bozen
Libera Università di Bolzano
Free University of Bozen · Bolzano

International Semantic Web Conference
12/10/MMXV

What are we going to learn today?

- How to organize and access your data using ontologies.
- How to do it with our system: –ontop–.
- How to use this approach for data integration and consistency checking.



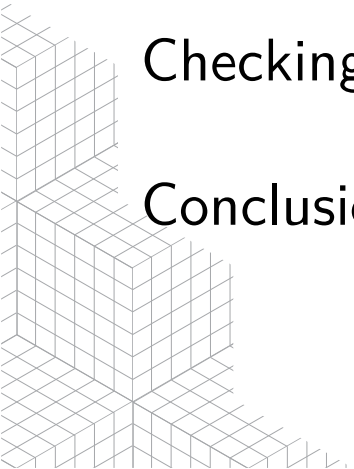
Overview

Ontology Based Data Access
- The Database:
- Ontologies
- Mappings
- Virtual Graph
- Querying

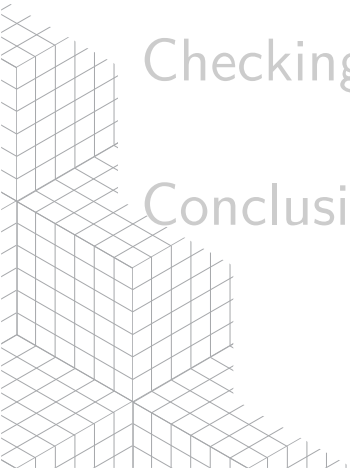
Ontology Based Data Integration
- SQL Federation

Checking Consistency

Conclusions

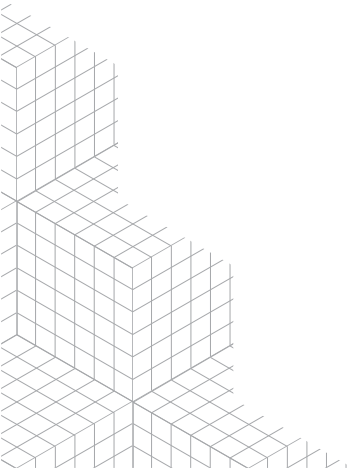


Outline



Simple Life Science Running Example

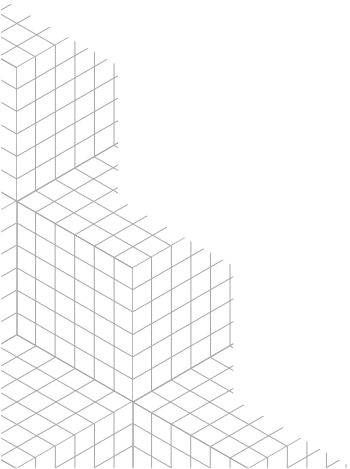
- Data source(s): Hospital Databases with cancer patients (First 1, then 2)
- Ontology: A common domain vocabulary defining Patient, Cancer, LungCancer, etc.
- Mappings: Relating the vocabulary and the databases.



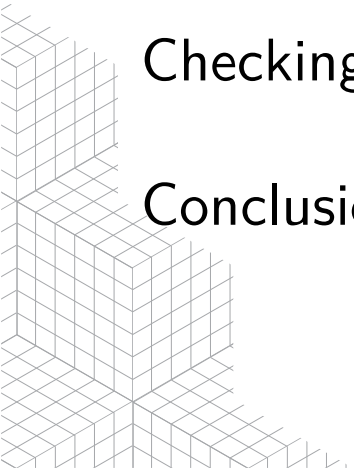
Pre-requisites

Before we start you need:

- Java
- The material online:
  `https://github.com/ontop/ontop-examples/tree/master/iswc-tutorial-2015`

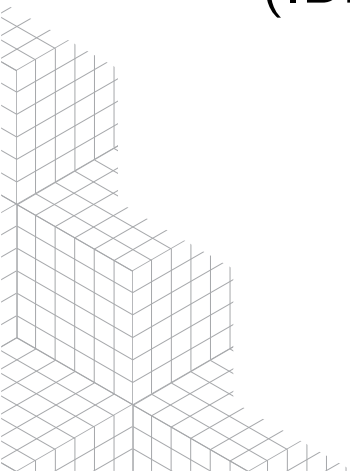


Outline



DB Engines and SQL

- Standard way to store LARGE volumes of data: Mature, Robust and FAST.
- Domain is structured as tables, data becomes rows in these tables.
- Powerful query language (SQL) to retrieve this data.
- Major companies developed SQL DBs for the last 30 years (IBM, Microsoft, Oracle)..



Cancer Patient Database 1
Table: tbl_patient

PatientId	Name	Type	Stage
1	Mary	false	4
2	John	true	7

Cancer Patient Database 1
Table: tbl_patient

PatientId	Name	Type	Stage
1	Mary	false	4
2	John	true	7

Type is:

- false for Non-Small Cell Lung Cancer (NSCLC)
- true for Small Cell Lung Cancer (SCLC)

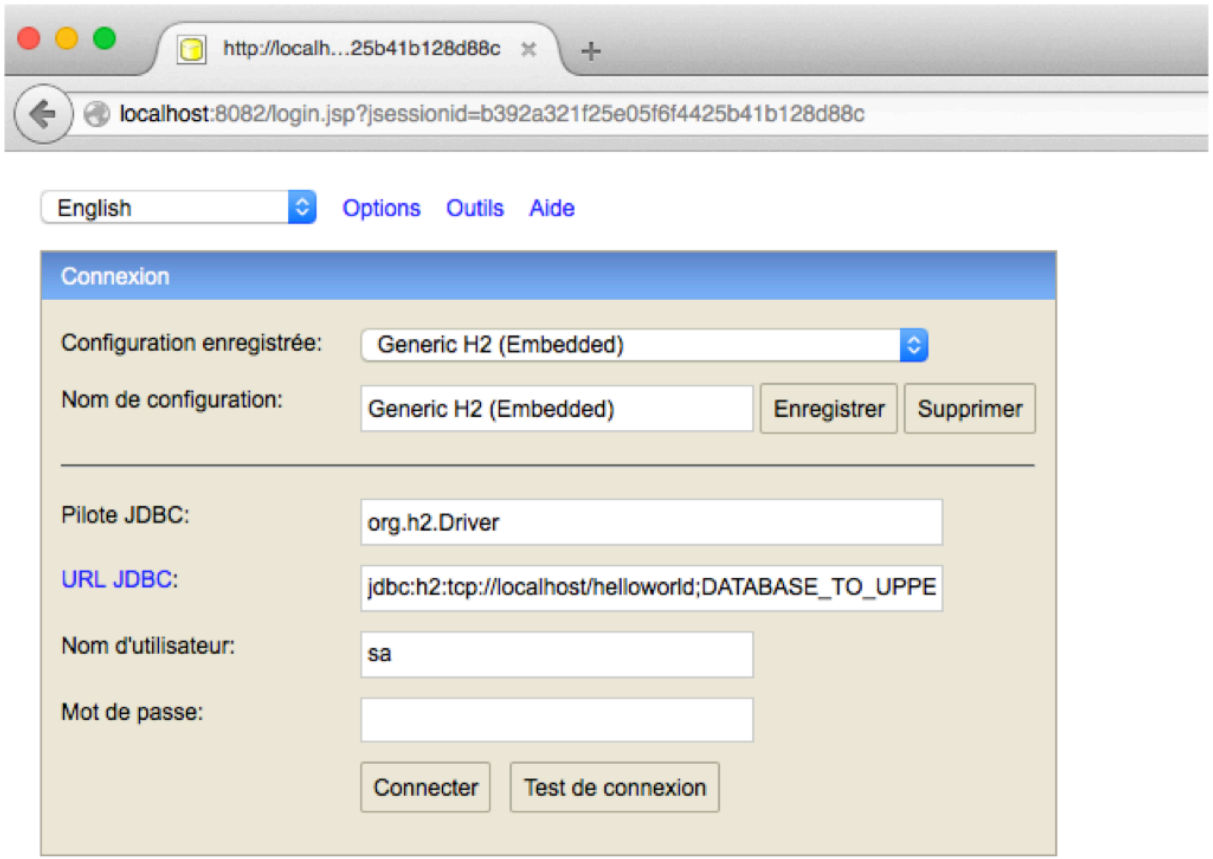
Stage is:

- 1-6 for NSCLC stages: I,II,III,IIIa,IIIb,IV
- 7-8 for SCLC stages: Limited,Extensive

Creating the DB in H2

- H2 is a pure java SQL database
- Just unzip the downloaded package
- Easy to run, just run the scripts:
  - Open a terminal (in mac Terminal.app, in windows run cmd.exe)
  - Move to the H2 folder (e.g., cd h2)
- Start H2 using the h2 scripts
  - sh h2.sh (in mac/linux - You might need "chmod u+x h2.sh ")
  - h2w.bat (in windows)

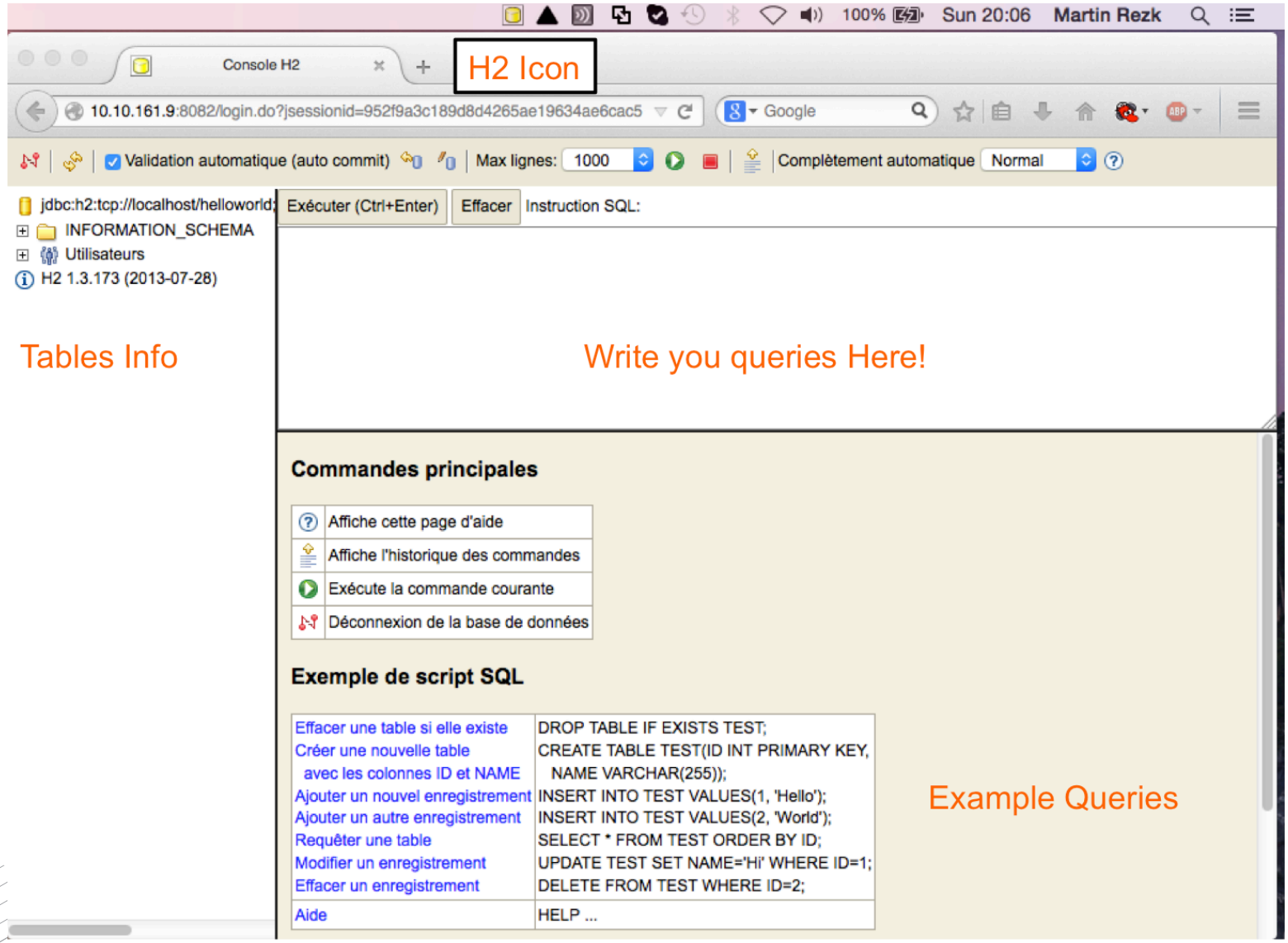
How it looks:



- jdbc:h2:tcp: = protocol information
- locahost = server location
- helloworld= database name

How to access it from the web

Optique



Creating the table

You can use the files create.sql and insert.sql

```
CREATE TABLE "tbl_patient" (
patientid INT NOT NULL PRIMARY KEY,
name VARCHAR(40),
type BOOLEAN,
stage TINYINT
)
```

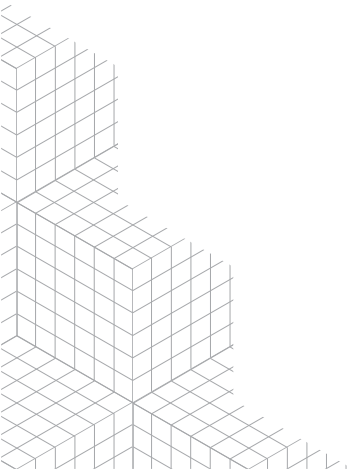
Adding Data:

```
INSERT INTO "tbl_patient"
(patientid,name,type,stage)
VALUES
(1,'Mary',false,4),
(2,'John',true,7);
```

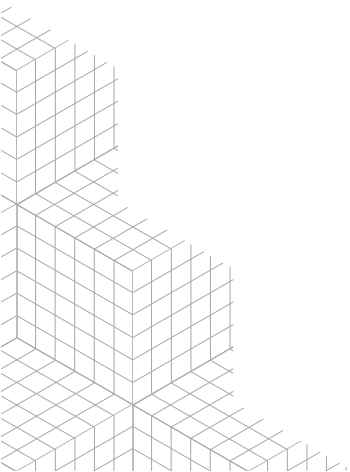
Example SQL Query

Patients with type false and stage IIIa or above (select.sql)

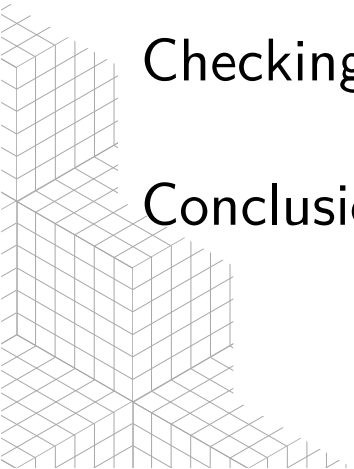
```
SELECT patientid
FROM "tbl_patient"
WHERE
TYPE = false AND stage >= 4
```



Give me the id and the name of the patients with a tumor at stage IIIa



Outline

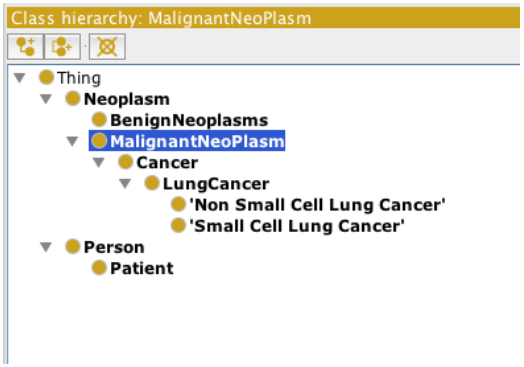


Ontologies: Conceptual view

Definition

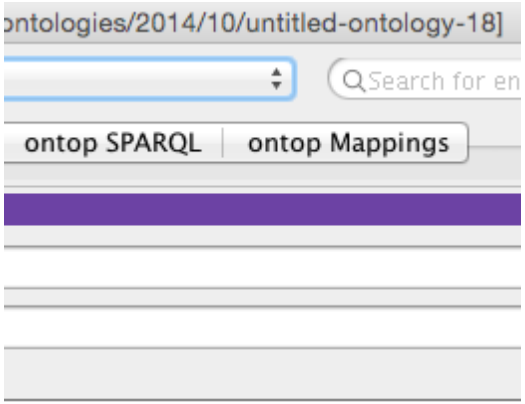
An artifact that contains a vocabulary, relations between the terms in the vocabulary, and that is expressed in a language whose syntax and semantics (meaning of the syntax) are shared and agreed upon.

Mike type Patient
NSCLC subClassOf LungCancer
LungCancer subClassOf Cancer



Ontologies: Protege

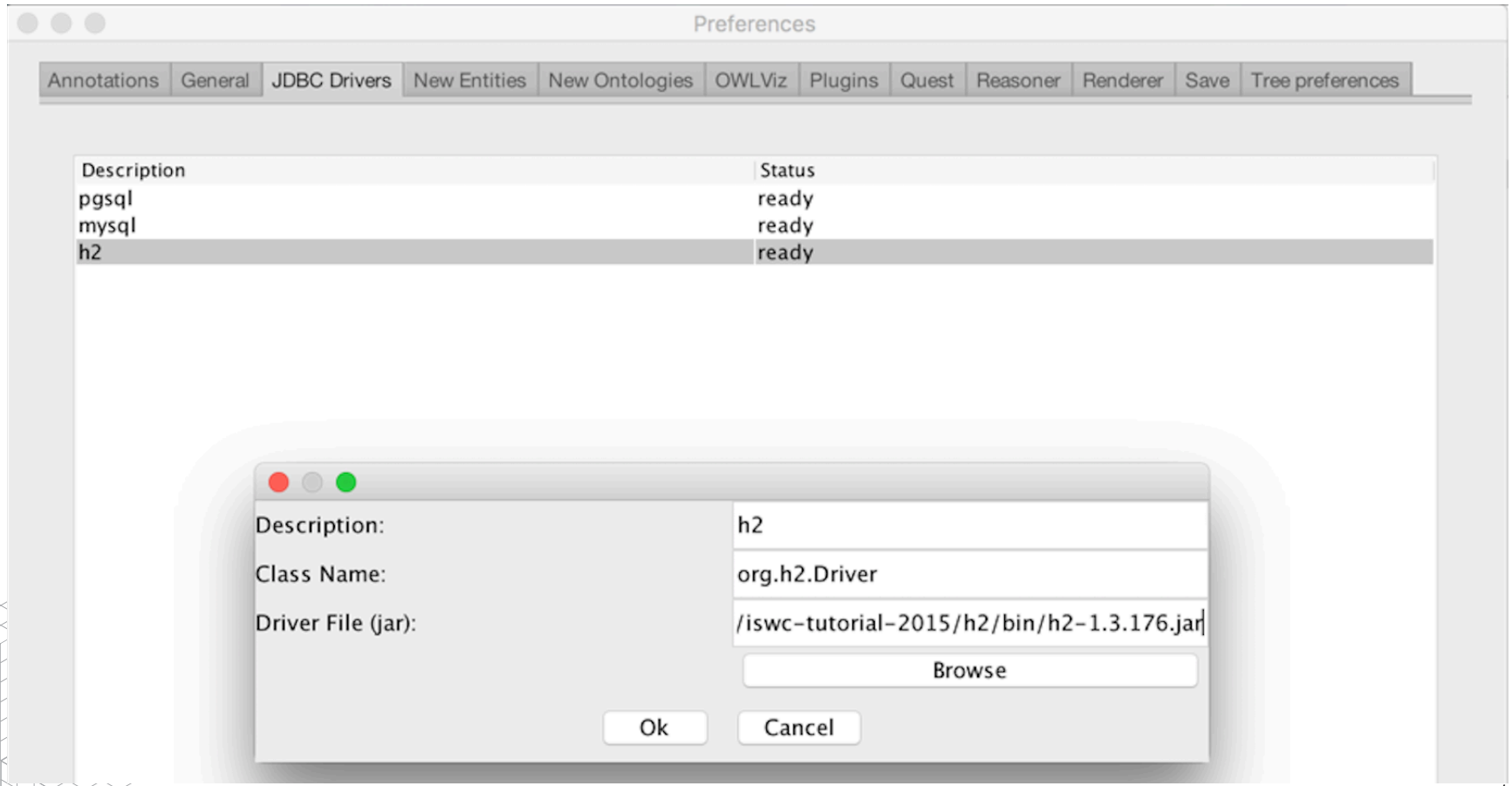
- Go to the protégé-ontop folder from your material. This is a Protégé 5 package that includes the ontop (1.15) plugin
- Run Protégé from the console using the run.bat or run.sh scripts. That is, execute:
- cd Protege_5/; run.sh



JDBC Driver configuration

Optique

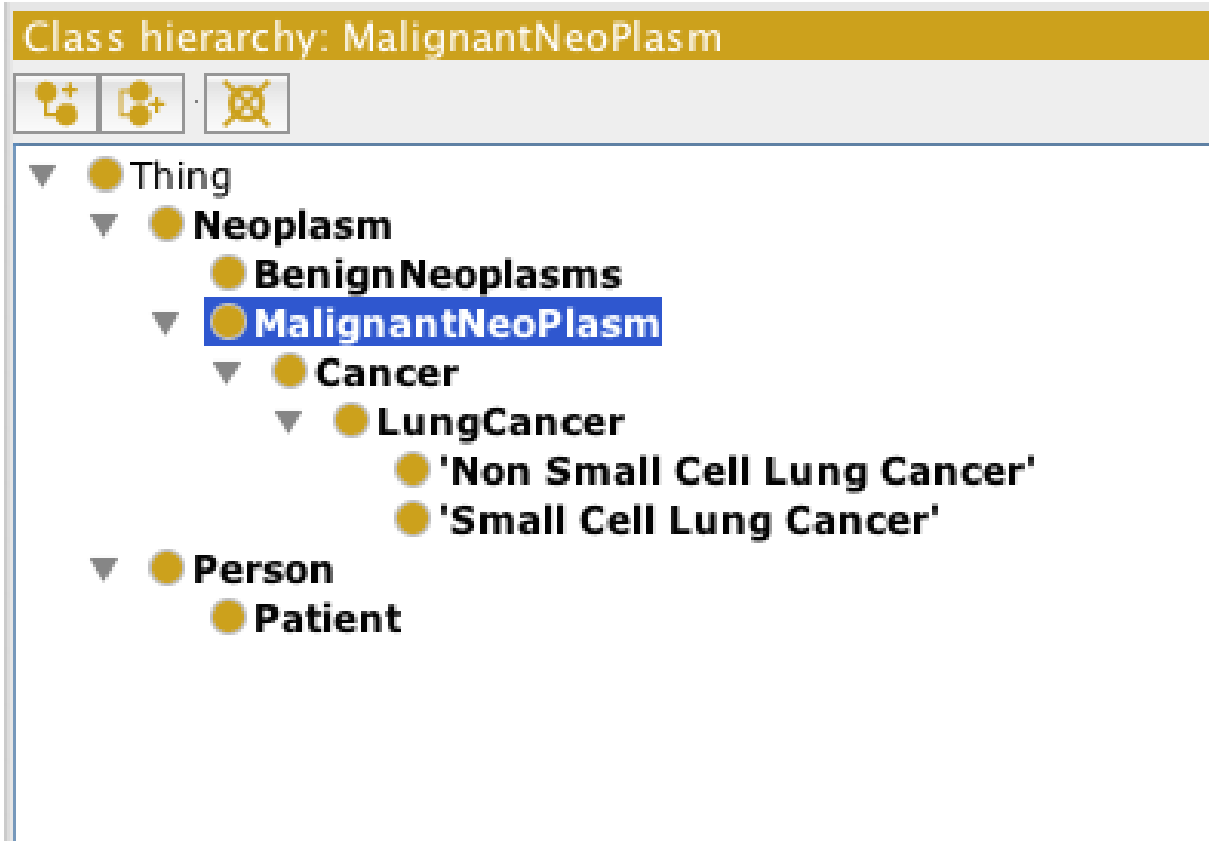
- Open “Preferences”, “JDBC Drivers” and add the configuration for H2
  - Description: `h2`
  - Class Name: `org.h2.Driver`
  - Driver File (jar): `/path/to/h2/bin/h2-1.3.176.jar`



The ontology: Creating Concepts and Properties

Optique

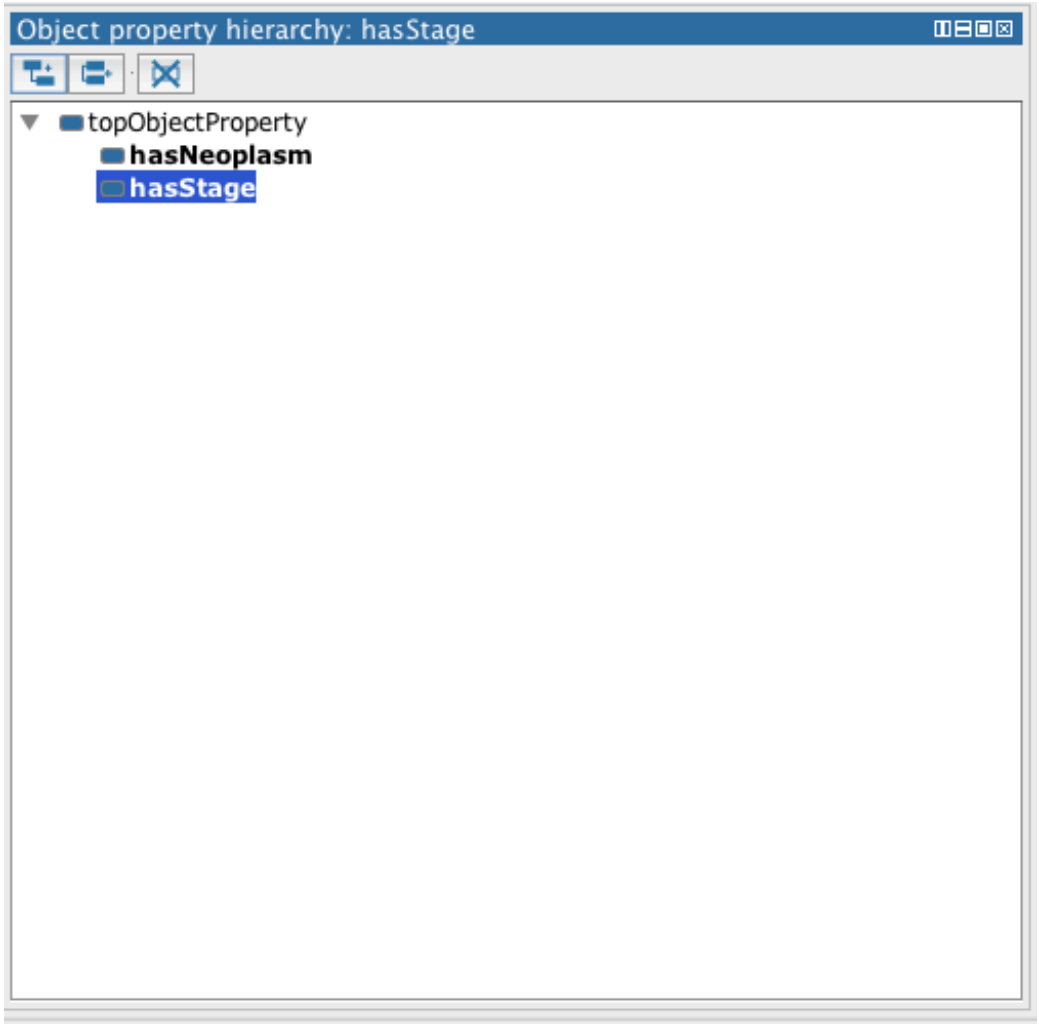
Add the concept: Patient.



(See PatientOnto.owl)

The ontology: Creating Concepts and Properties

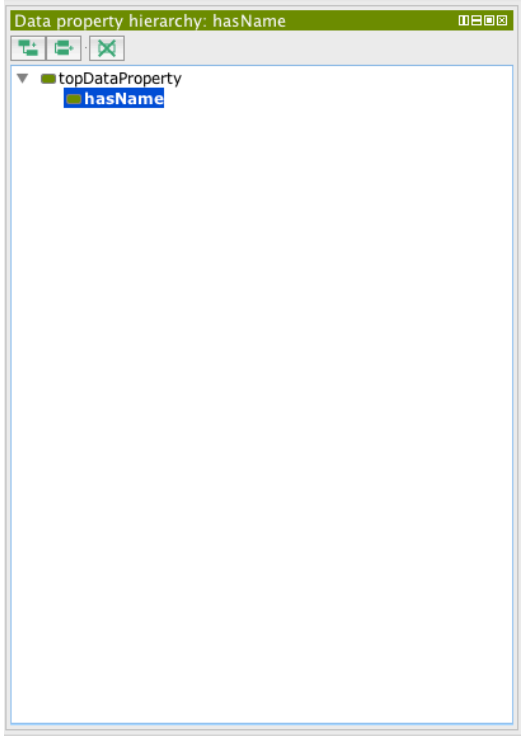
Add these object properties:



The ontology: Creating Concepts and Properties

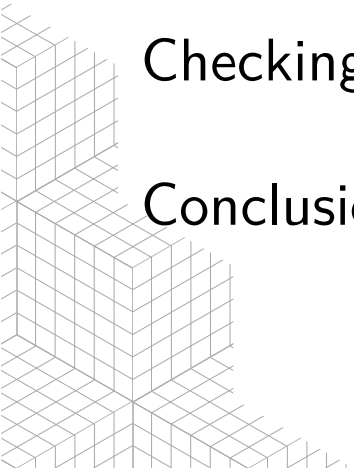
Optique

Add these data properties:



(See PatientOnto.owl)

Outline



Mappings

- We have the vocabulary, the database, now we need to link those two.
- Mappings define triples (subject, property, object) out of SQL queries.
- These triples is accessible during query time (the on-the-fly approach) or can be imported into the OWL ontology (the ETL approach)

Definition (Intuition)

A mapping have the following form:

$$\textit{TripleTemplate} \quad \leftarrow \text{SQL Query to build the triples}$$

represents triples constructed from each result row returned by the SQL query in the mapping.

The Mappings

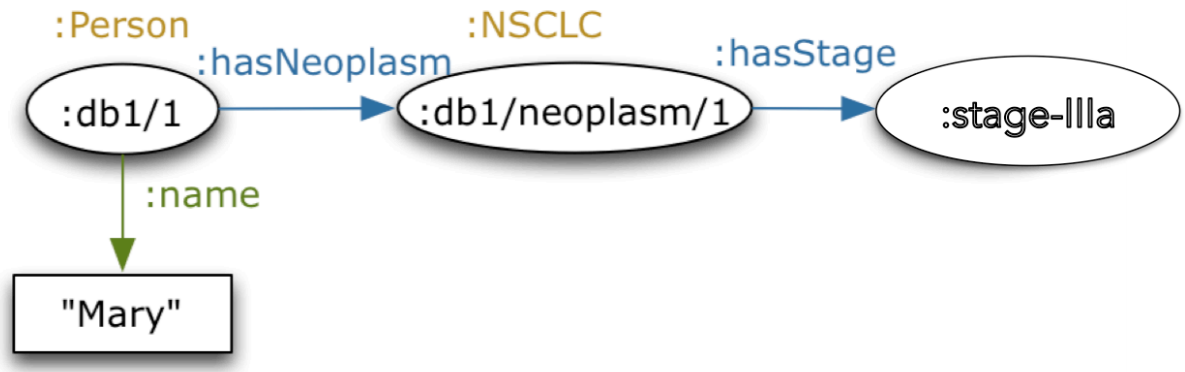
p.Id	Name	Type	Stage
1	Mary	false	2

- (:db1/{p.id},type, :Patient) ← Select p.id From tbl_patient
- (:db1/{p.id},:hasName, {name}) ← Select p.id,name From tbl_patient
- (:db1/{p.id},:hasNeoplasm, :db1/neoplasm/{p.id}) ← Select p.id From tbl_patient
- (:db1/neoplasm/{p.id},:hasStage, :stage-IIIa) ← Select p.id From tbl_patient where stage=4

The Mappings

p.Id	Name	Type	Stage
1	Mary	false	2

- (:db1/{p.id},type, :Patient) ← Select p.id From tbl_patient
- (:db1/{p.id},:hasName, {name}) ← Select p.id,name From tbl_patient
- (:db1/{p.id},:hasNeoplasm, :db1/neoplasm/{p.id}) ← Select p.id From tbl_patient
- (:db1/neoplasm/{p.id},:hasStage, :stage-IIIa) ← Select p.id From tbl_patient where stage=4



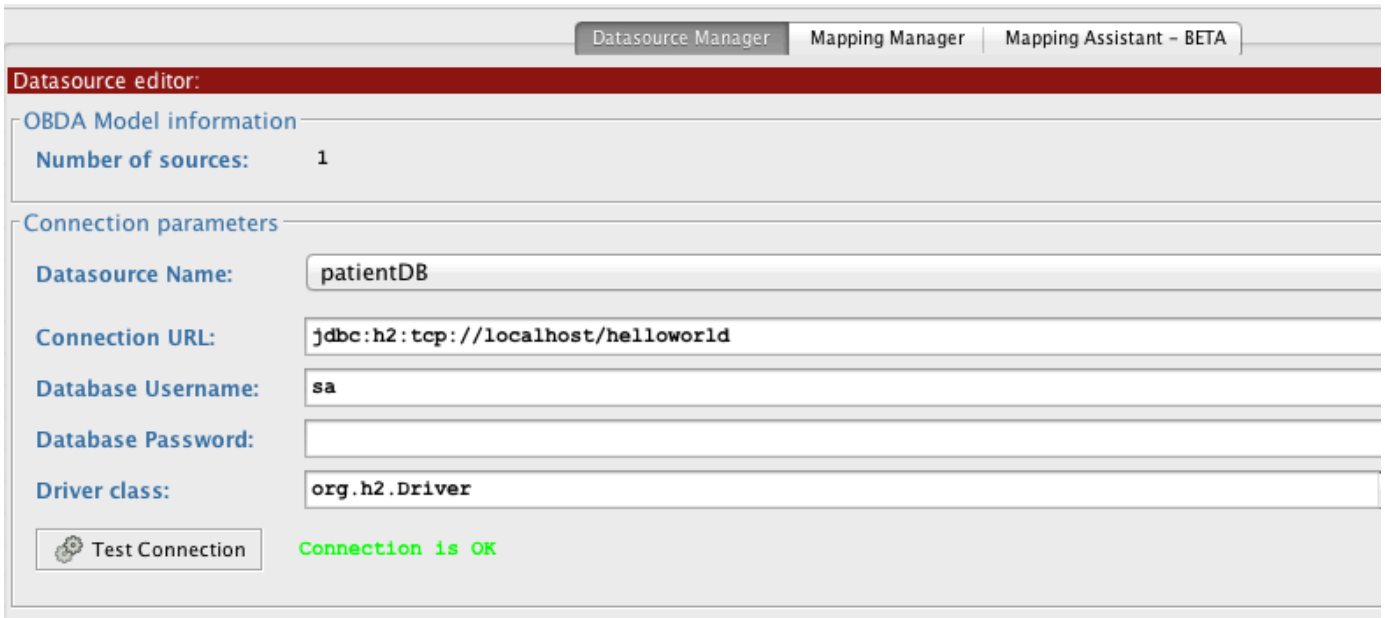
The Mappings

Using the Ontop Mapping tab, we now need to define the connection parameters to our lung cancer database
Steps:

1. Switch to the Ontop Mapping tab
2. Add a new data source (give it a name, e.g., PatientDB)
3. Define the connection parameters as follows:
   - Connection URL: jdbc:h2:tcp://localhost/helloworld
   - Username: sa
   - Password: (leave empty)
   - Driver class: org.h2.Driver (choose it from the drop down menu)
4. Test the connection using the "Test Connection" button

The Mappings

Optique



The Mappings

- Switch to the "Mapping Manager" tab in the ontop mappings tab.
- Select your datasource
- click Create:

```
target:  :db1/{patientid} a :Patient .
source:  SELECT patientid FROM "tbl_patient"
target:  :db1/{patientid} :hasName {name} .
source:  Select patientid,name FROM "tbl_patient"
target:  :db1/{patientid} :hasNeoplasm :db1/neoplasm/{patientid}.
source:  SELECT patientid FROM "tbl_patient"
 target:  :db1/neoplasm/{patientid} :hasStage :stage-IIIa .
source:  SELECT patientid FROM "tbl_patient" where stage=4
```

The Mappings

- Now we classify the neoplasm individual using our knowledge of the database.
- We know that “false” in the table patient indicates a “Non Small Cell Lung Cancer”, so we classify the patients as a :NSCLC.

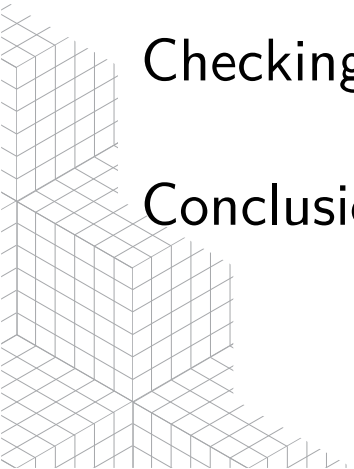
nsclc

```
:db1/neoplasm/{PATIENTID}/ a :NSCLC .
select * from TBL_PATIENT WHERE TYPE = false
```

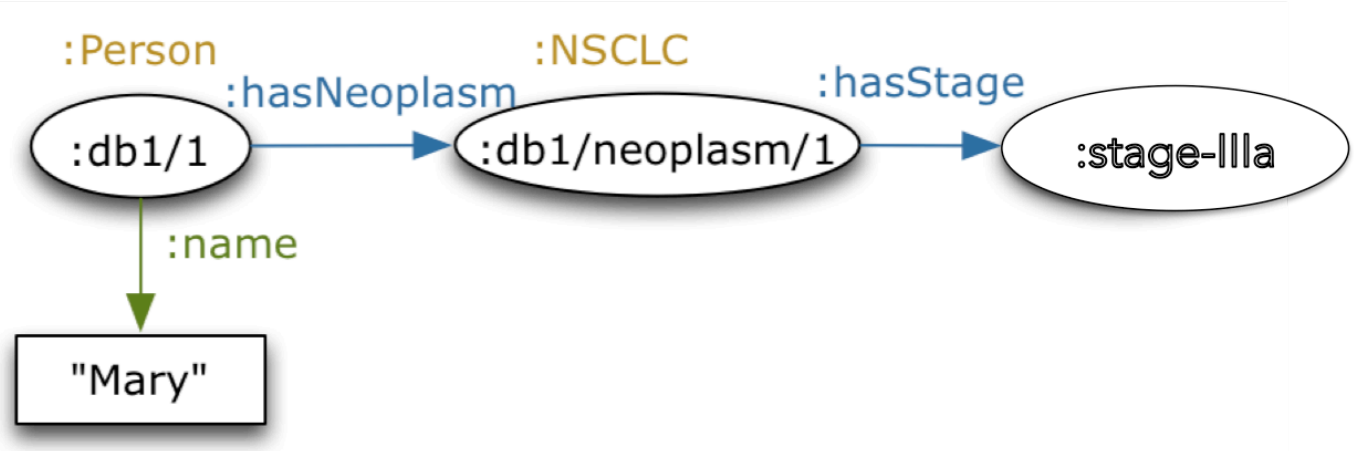
sclc

```
:db1/neoplasm/{PATIENTID}/ a :SCLC .
select * from TBL_PATIENT WHERE TYPE = true
```

Outline

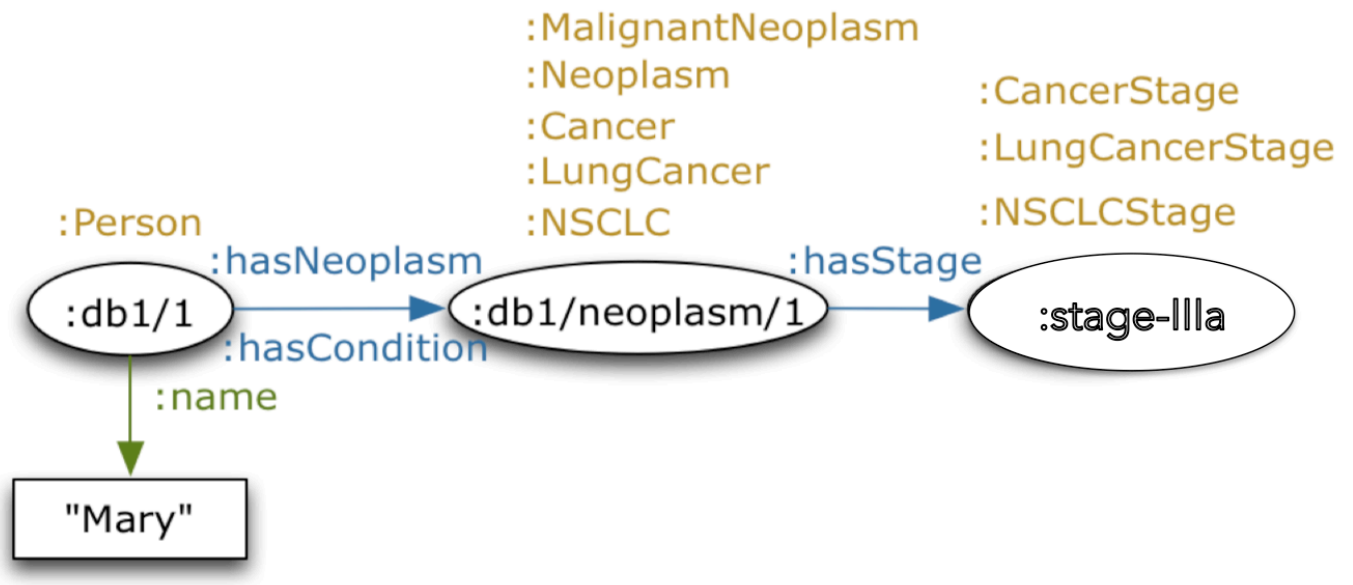


Data:



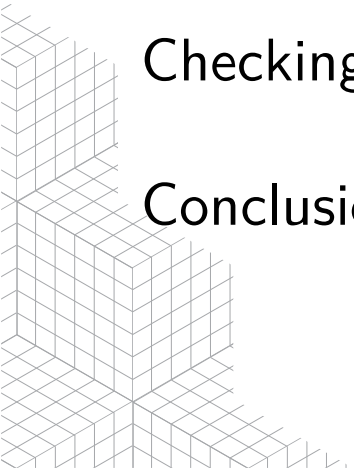
- The vocabulary is more domain oriented and independent from the DB.
- No more values to encode types or stages.
- Later, this will allow us to easily integrate new data or domain information (e.g., an ontology).
- Our data sources are now documented!.

Data and Inference:



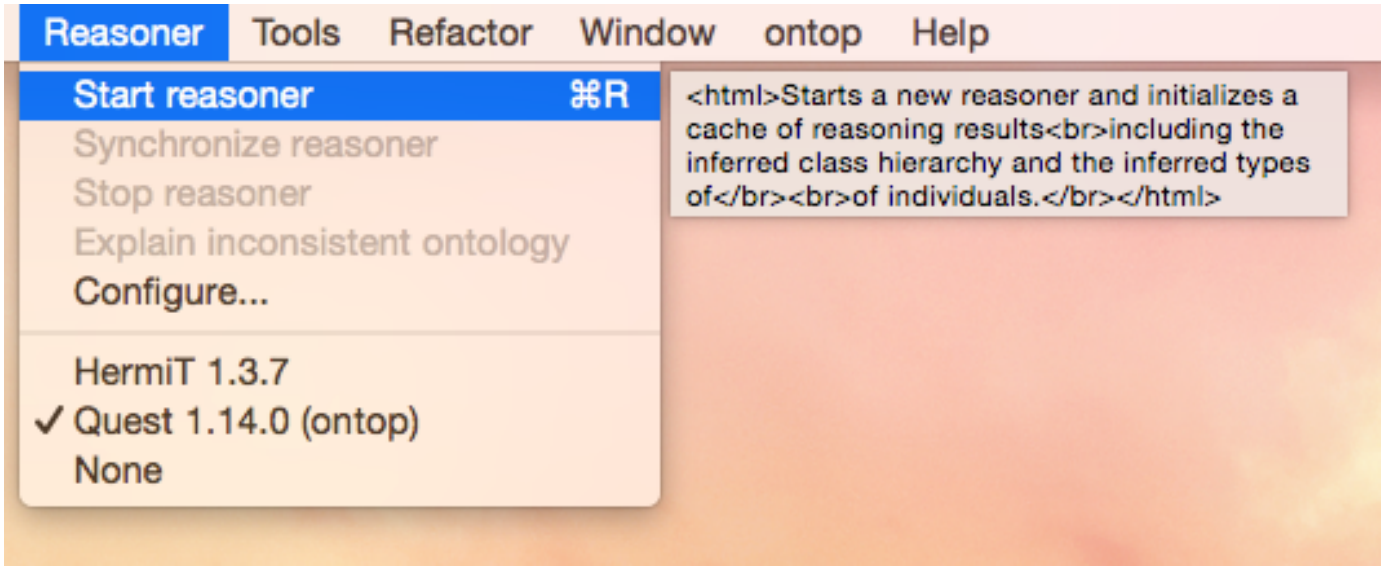
- There is a new individual :db1/neoplasm/1 that stands for the cancer (tumor) of Mary. This allows the user to query specific properties of the tumor independently of the patient.
- We get extra information as shown above.

Outline



On-the-fly access to the DB

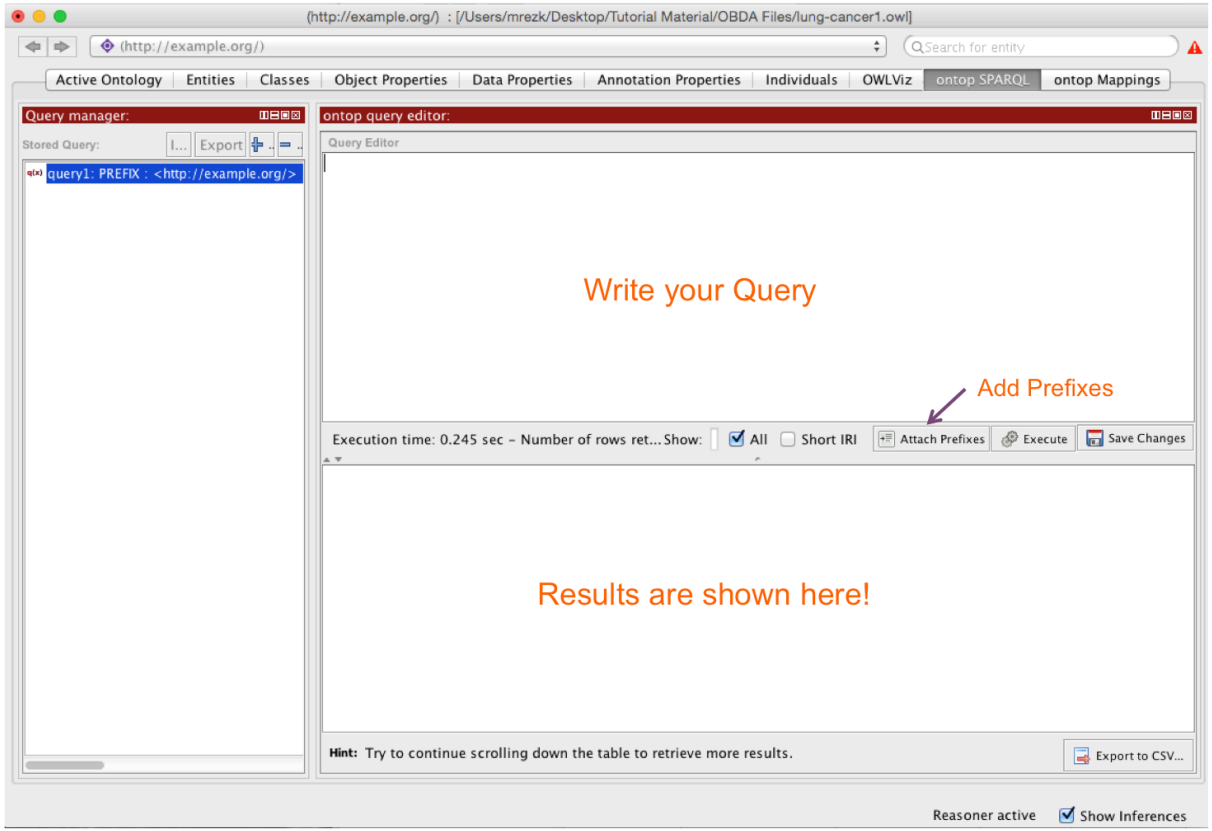
- Recall our information need: Give me the id and the name of the patients with a tumor at stage IIIa.
- Enable Ontop in the "Reasoner" menu



On-the-fly access to the DB

Optique

In the ontop SPARQL tab add all the prefixes



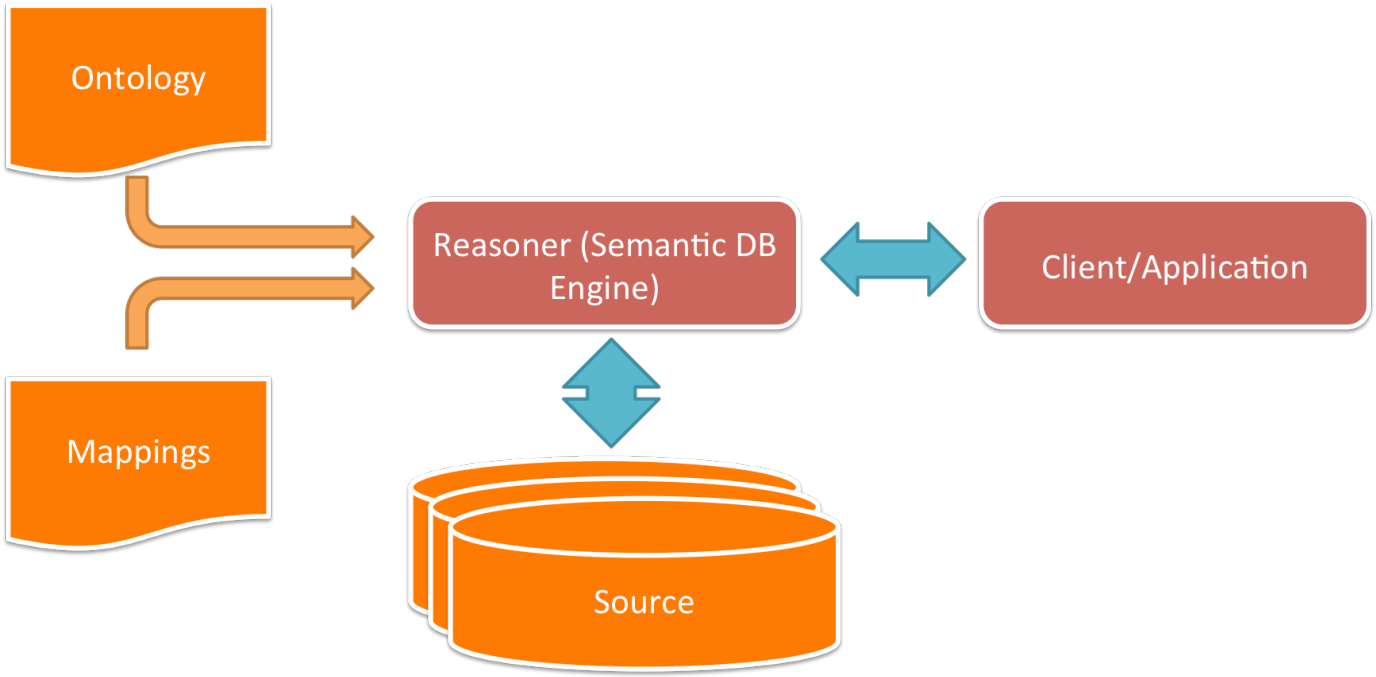
On-the-fly access to the DB

- Write the SPARQL Query

```
SELECT ?p ?name WHERE
{ ?p rdf:type :Patient .
 ?p :hasName ?name .
 ?p :hasNeoplasm ?tumor .
 ?tumor :hasStage :stage-IIIa .}
```

- Click execute
- This is the main way to access data in ontop and its done by querying ontop with SPARQL.

How we do Inference...



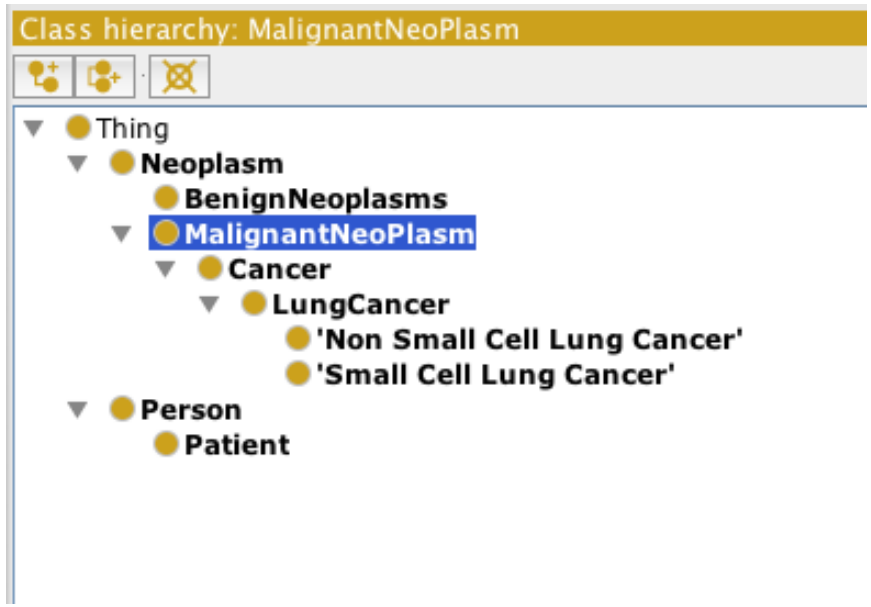
We embed inference into the query
We do not need to reason with the (Big) Data

Where Reasoning takes Place

Optique

- If we pose the query asking for all the instances of the class Neoplasm:

SELECT ?x WHERE { ?x rdf:type :Noeplasms). }



Where Reasoning takes Place

- If we pose the query asking for all the instances of the class Neoplasm:

```
SELECT ?x WHERE { ?x rdf:type :Noeplasms). }
```

- (Intuitively) -ontop- will translate it into:

```
SELECT ?x WHERE { { ?x rdf:type :Neoplasms. }
                  UNION
                  { ?x rdf:type :BenignNeoplasms. }
                  UNION
                  { ?x rdf:type :MalignantNeoplasm. }
                  UNION
                  .
                  .
                  .
                  { ?x rdf:type :NSCLC). }
                  UNION
                  { ?x rdf:type :SCLC). } }
```

Where Reasoning takes Place

Optique

- If we pose the query asking for all the instances of the class Neoplasm:

```
SELECT ?x WHERE { ?x rdf:type :Noeplasms). }
```

- (Intuitively) -ontop- will translate it into:

```
SELECT ?x WHERE { {?x rdf:type :Neoplasms.}
                  UNION
                  { ?x rdf:type :BenignNeoplasms. }
                  UNION
                  { ?x rdf:type :MalignantNeoplasm . }
                  UNION
                  .
                  .
                  { ?x rdf:type :NSCLC). }
                  UNION
                  { ?x rdf:type :SCLC). } }
```

Where Reasoning takes Place

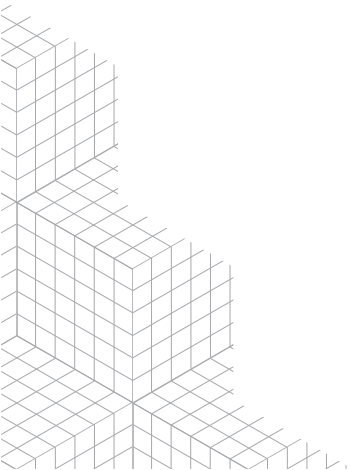
- If we pose the query asking for all the instances of the class Neoplasm:

```
SELECT ?x WHERE { ?x rdf:type :Noeplasms). }
```

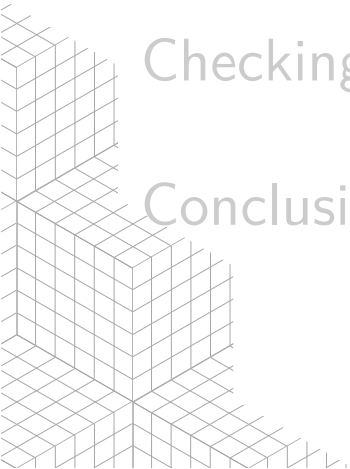
- (Intuitively) -ontop- will translate it into:

```
SELECT Concat(:db1/neoplasm/, TBL.PATIENT.id) AS ?x
FROM TBL.PATIENT
```

- Ontology languages: RDF, RDFS, OWL (W3C recommendations)
- User Query Language: SPARQL (W3C recommendation)
- Mappings: R2RML (W3C recommendation)
- DB Query Language: SQL



Outline



What about Data Integration?

Optique

Cancer Patient Database 2

T_Name

PId	Nombre
1	Anna
2	Mike

DB information is distributed in multiple tables. The IDs of the two DBs overlap.

T_NSCLC

Id	hosp	Stge
1	X	two
2	Y	one

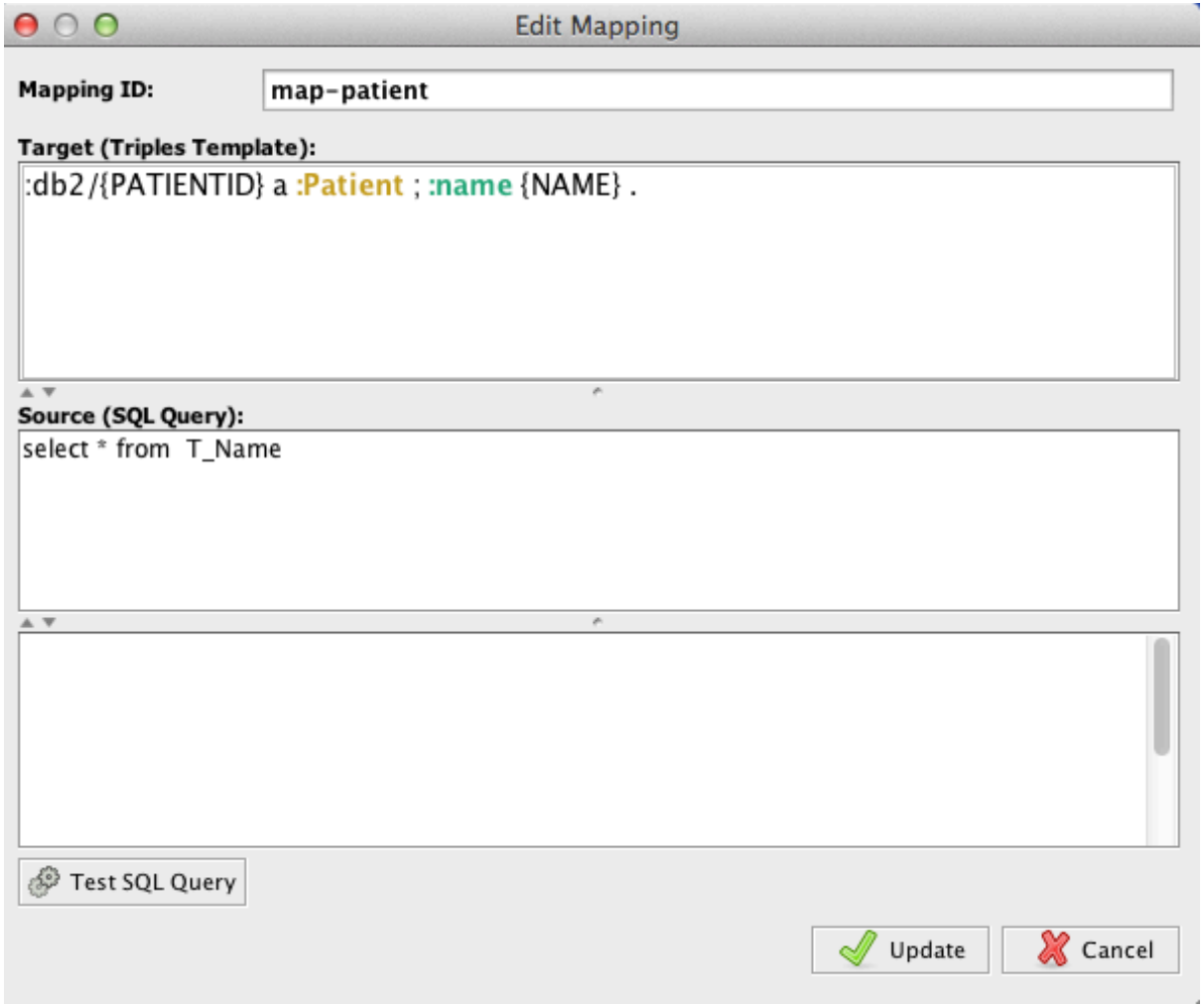
Information is encoded differently. E.g. Stage of cancer is text (one, two...)

T_SCLC

key	hosp	St
1	XXX	
2	YYY	

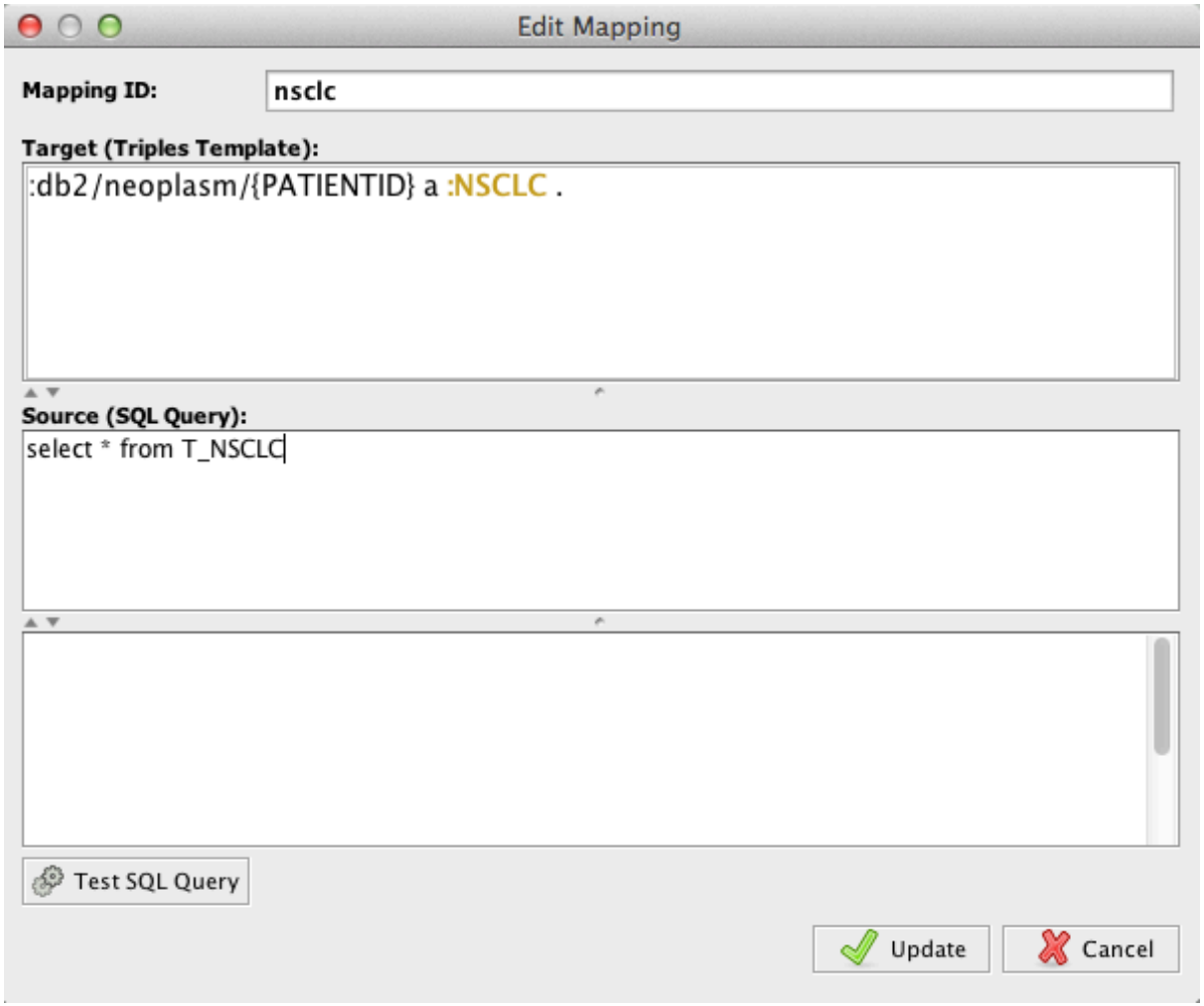
New Mappings

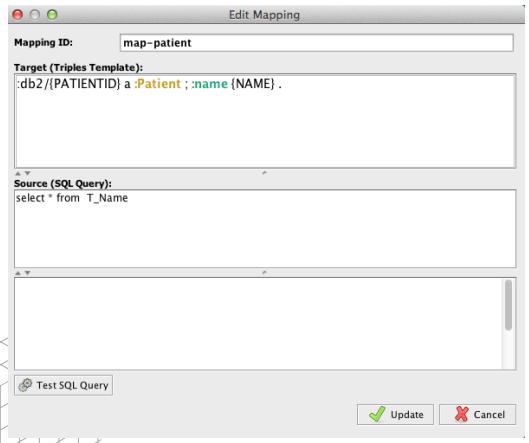
Optique



New Mappings

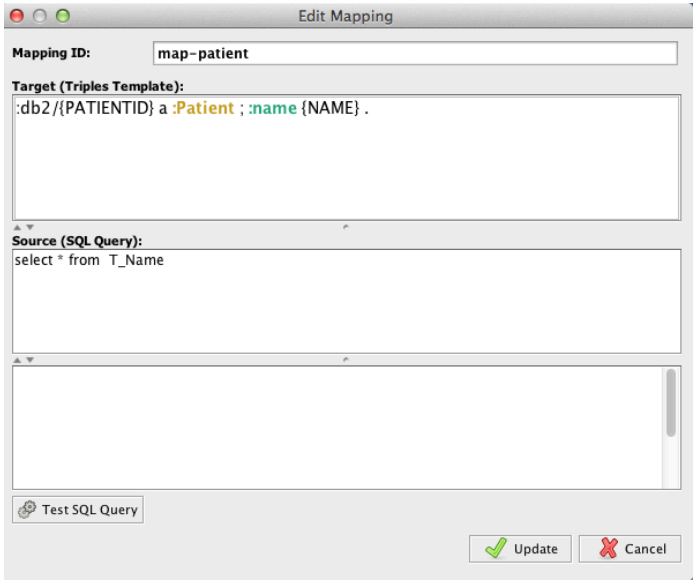
Optique

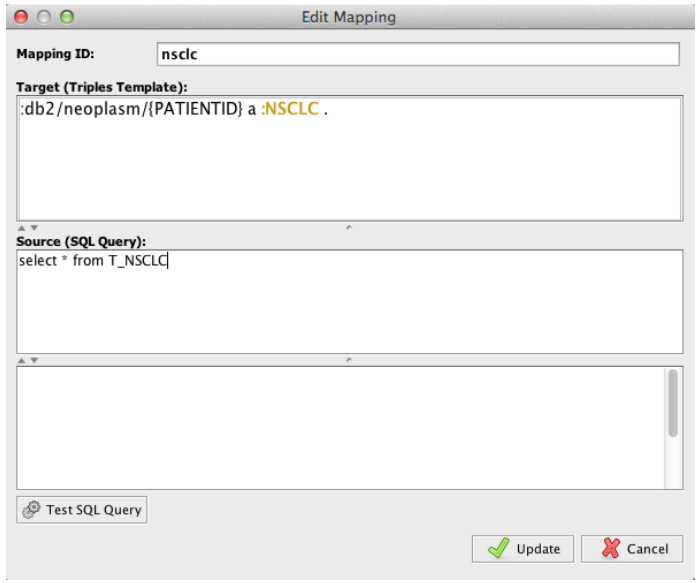




New Mappings

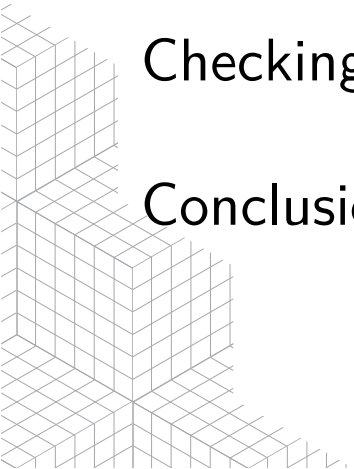
Optique





- The URI's for the new individuals differentiate the data sources (db2 vs. db1)
- Being an instance of NSCLC and SCLC depends now on the table, not a column value

Outline

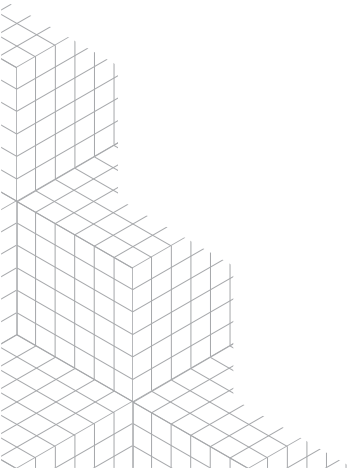


Federation: Exareme

Optique

`http://www.exareme.org`

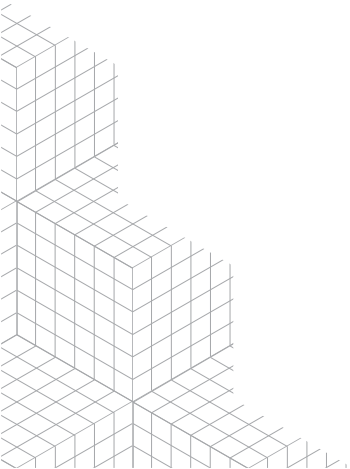
- Developed at the University of Athens.
- Query Processing module of the Optique Platform.
- Soon it will be release as Open Source.



Federation: Exareme

Steps to make it work:

- Install Exareme (Developed at TUA).
  - You will need Python 2.7 and APSW.
- Build the JDBC URL for ontop. It consists of:
  - 1 fragment for the server where Exareme is installed
  - 1 fragment for each underlying DB.



Federation: Exareme

Steps to make it work:

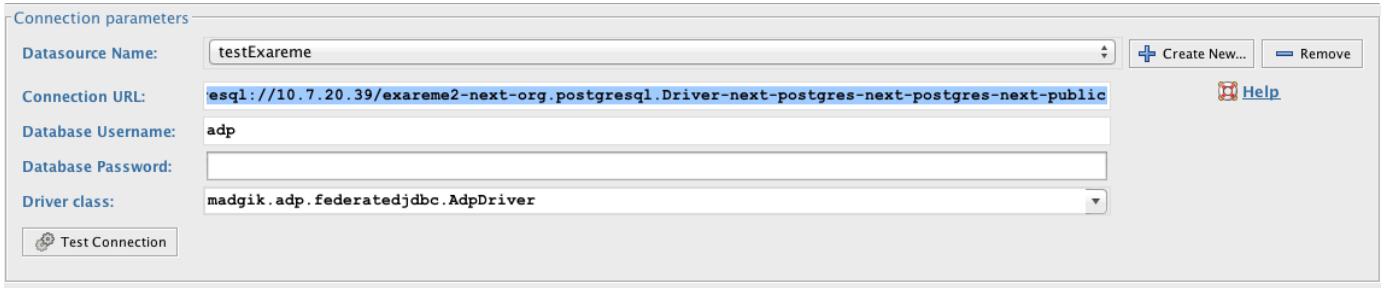
- Install Exareme (Developed at TUA).
  - You will need Python 2.7 and APSW.
- Build the JDBC URL for ontop. It consists of:
  - 1 fragment for the server where Exareme is installed
  - 1 fragment for each underlying DB.

```
jdbc:fedadp:http://10.7.20.80:9090/tmp-
fedDB-data1-next-jdbc:postgresql://10.7.20.80/exareme1-next-
org.postgresql.Driver-next-postgres-next-postgres-next-public-
fedDB-data2-next-jdbc:postgresql://10.7.20.39/exareme2-next-
org.postgresql.Driver-next-postgres-next-postgres-next-public
```

Federation: Exareme

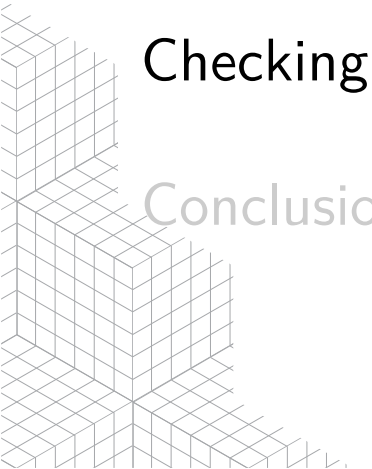
Steps to make it work:

- Install Exareme (Developed at TUA).
  - You will need Python 2.7 and APSW.
- Build the JDBC URL for ontop. It consists of:
  - 1 fragment for the server where Exareme is installed
  - 1 fragment for each underlying DB.



You can build your mappings and query as usual !!

Outline

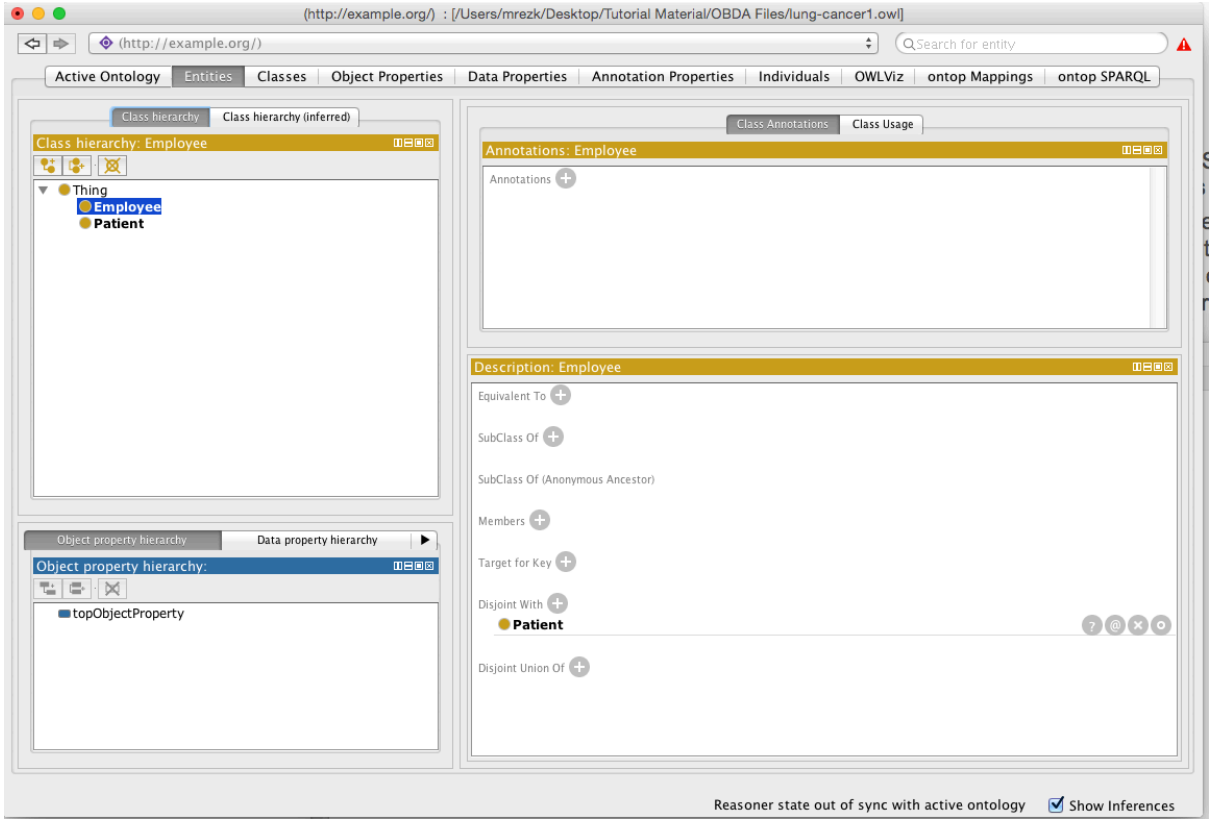


Consistency

- A logic based ontology language, such as OWL, allows ontologies to be specified as logical theories, this implies that it is possible to constrain the relationships between concepts, properties, and data.
- In OBDA inconsistencies arise when your mappings violate the constraints imposed by the ontology.
- In OBDA we have two types of constraints:
  - Disjointness: The intersection between classes Patient and Employee should be empty. There can also be disjoint properties.
  - Functional Properties: Every patient has at most one name.

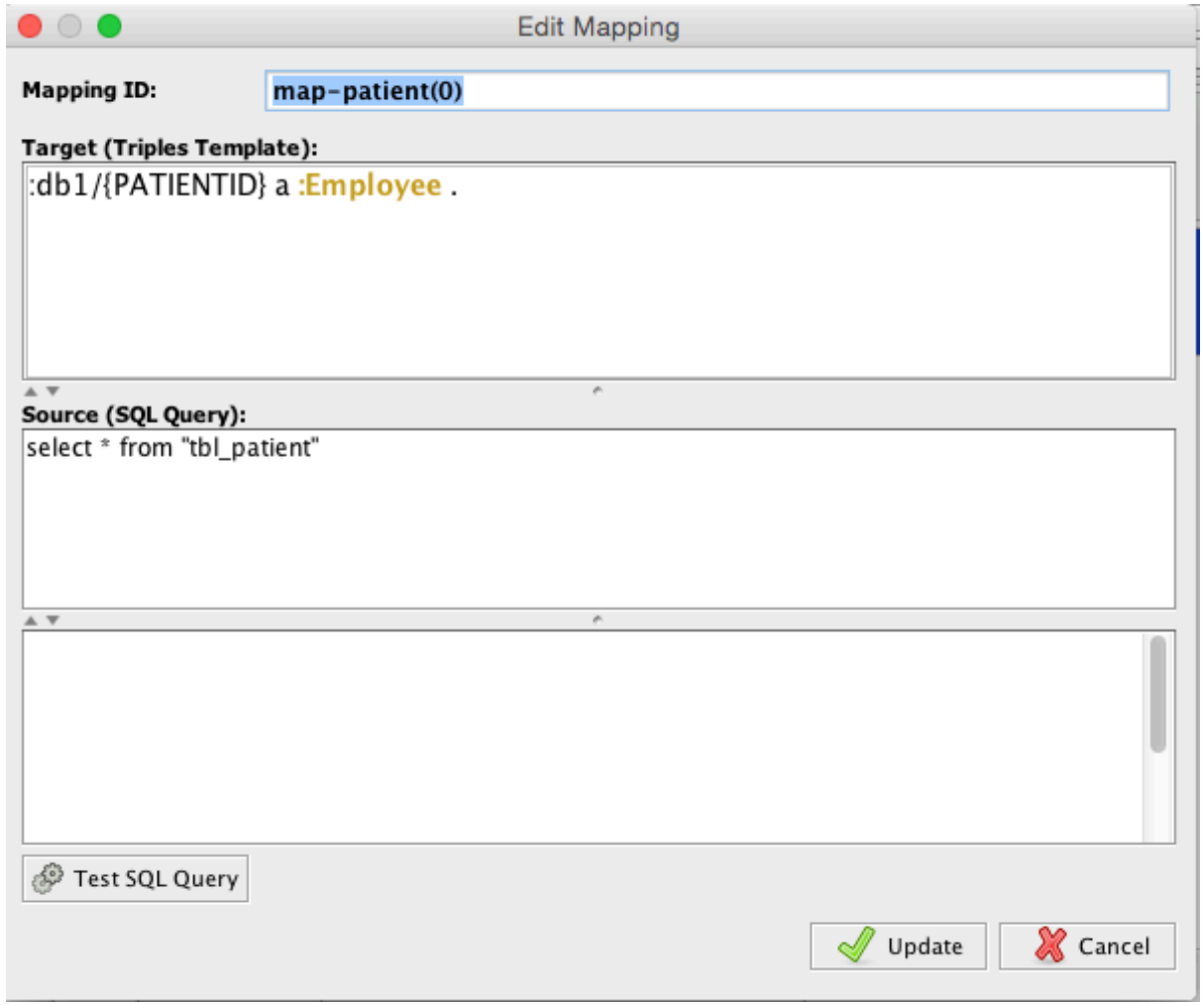
Consistency: Setting up a Constraint

Optique

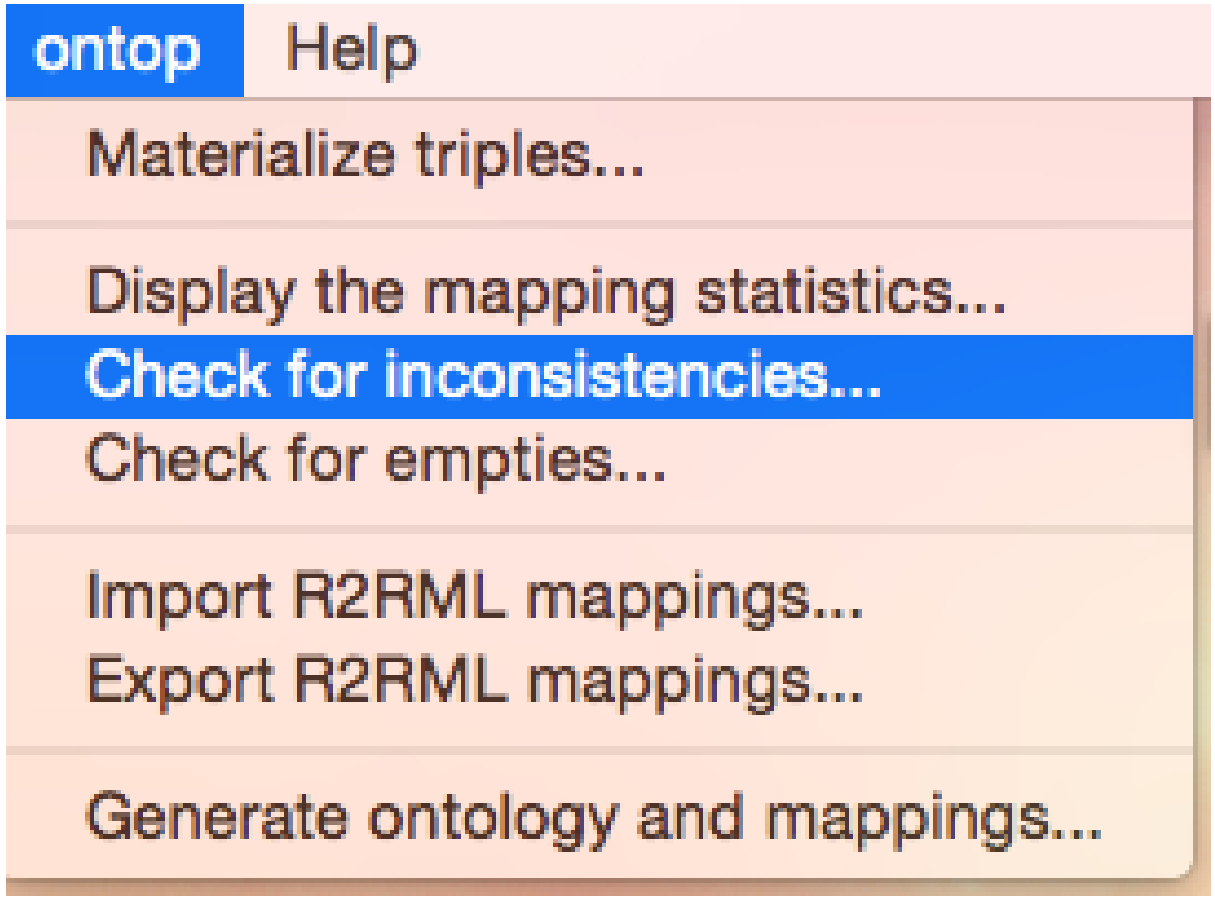


Consistency: Building a wrong mapping

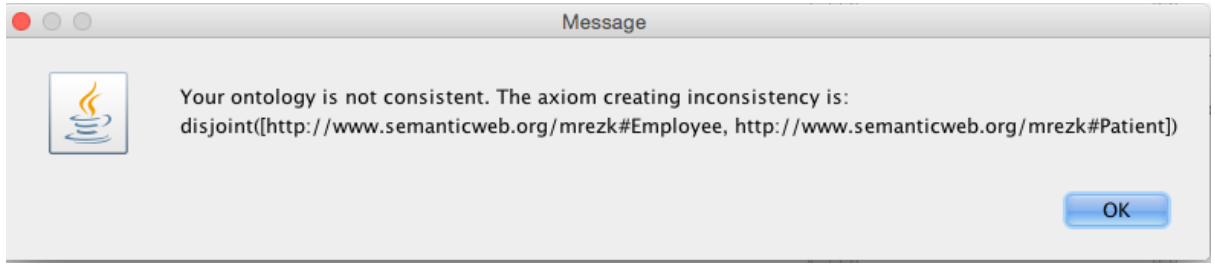
Optique



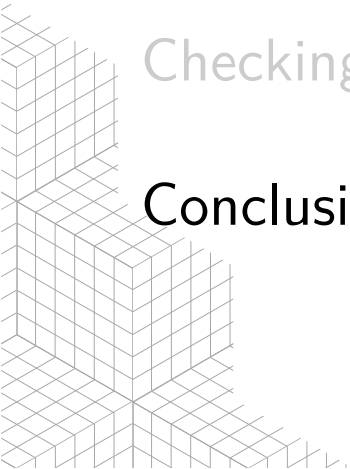
Consistency: Checking Inconsistency



Consistency: Finding out the Problem



Outline



Conclusions

- Ontologies give you a common vocabulary to formulate the queries, and mappings to find the answers.
- Ontologies and Semantic Technology can help to handle the problem of accessing Big Data
  - Diversity:
    - Using ontologies describing particular domains allows to hide the storage complexity.
    - Agreement on data identifiers allows for integration of datasets.
  - Understanding: Agreement on vocabulary allow to better define your data and allows for easy information exchange.
- There is no need of computationally expensive ETL processes.
- Reasoning is scalable because we reason at the query level.
- You do not need to have everything ready to use it!

What I left outside this talk...

- Semantic Query Optimisation
- SWRL and Recursion
- Performance Evaluation
- Aggregates and bag semantics
- Give out about Database Engines
- Tons of theory
- etc. etc. etc...

Thanks!!!

Visit us!

http://ontop.inf.unibz.it

http://www.optique-project.eu