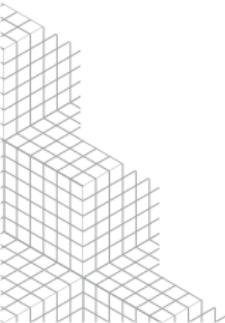


# Ontologies Ontop Databases

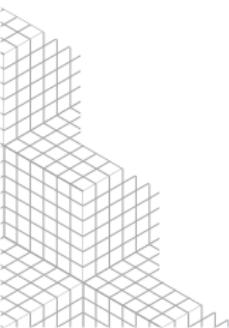
Martin Rezk (PhD)  
Faculty of Computer Science  
Free University of Bozen-Bolzano, Italy



Semantic Web Applications and Tools for Life Sciences  
9/12/MMXIV



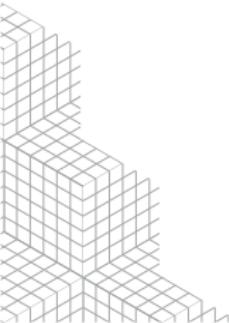
- Postdoc at: Free University of Bozen-Bolzano, Bolzano, Italy.
- Research topics: Ontology-based Data Access (OBDA), Efficient Query Answering, Query rewriting, Data integration.
- Leader of the -ontop- project.



# What are we going to learn today?

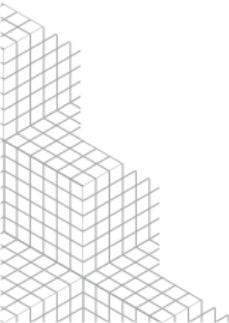
Optique

- 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**.



Part of the material here has been taken from a tutorial by  
Mariano Rodriguez (2011)

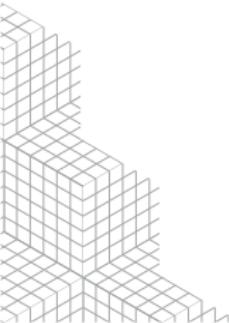
<http://www.slideshare.net/marianomx/ontop-a-tutorial>



## License

This work is supported by the Optique Project and is licensed under the Creative Commons Attribution-Share Alike 3.0 License

<http://creativecommons.org/licenses/by-sa/3.0/>



## Introduction: Optique and Ontop

### Ontology Based Data Access

The Database:

Ontologies

Mappings

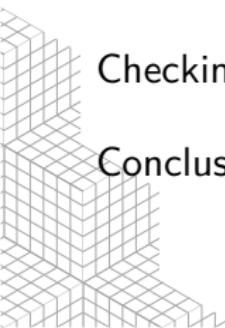
Virtual Graph

Querying

### Data Integration

### Checking Consistency

### Conclusions



## Introduction: Optique and Ontop

### Ontology Based Data Access

The Database:

Ontologies

Mappings

Virtual Graph

Querying

### Data Integration

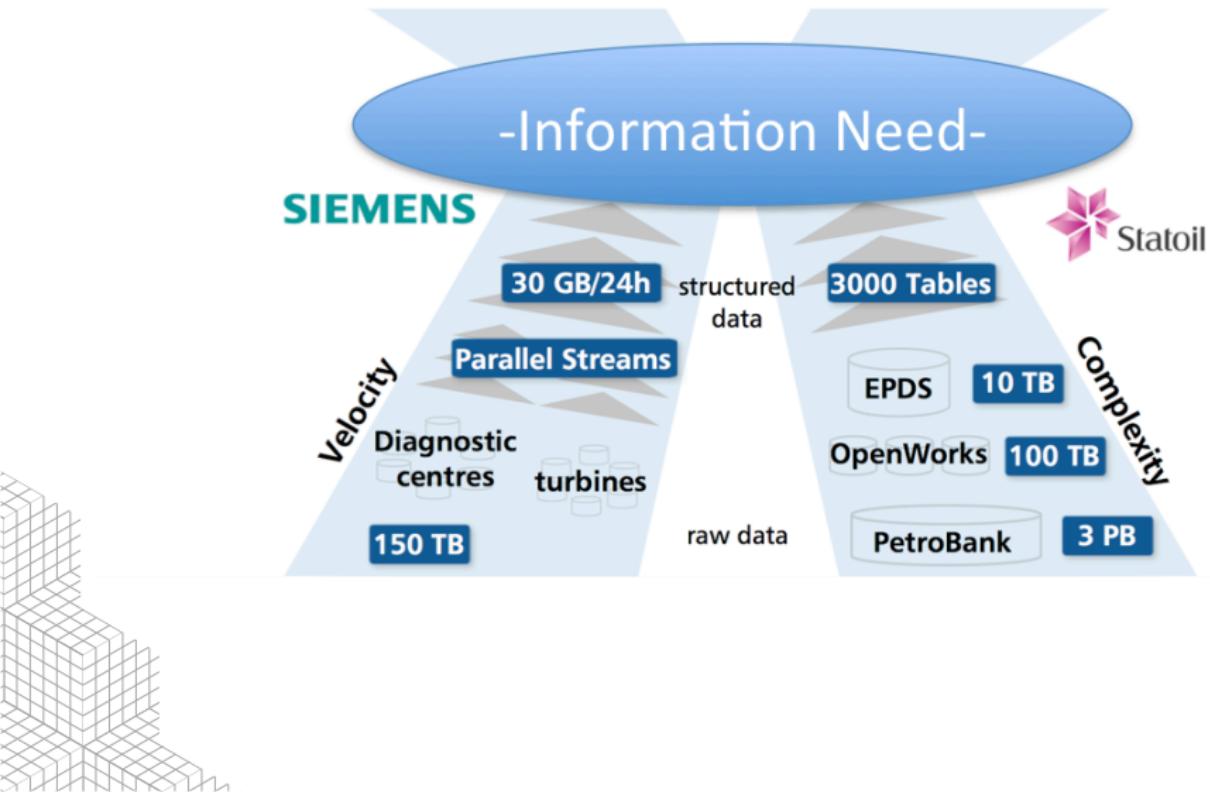
### Checking Consistency

### Conclusions



# Ontop and Optique: Why?

Optique



# When does this **Go Wrong?**

Optique



I'm sure the information is there but there are so many concepts involved that I **can't find it in the application**.

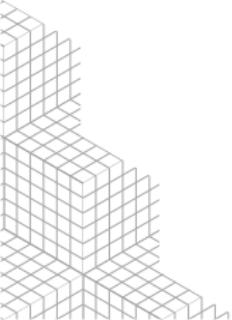


I **can't say what I want** to the application.



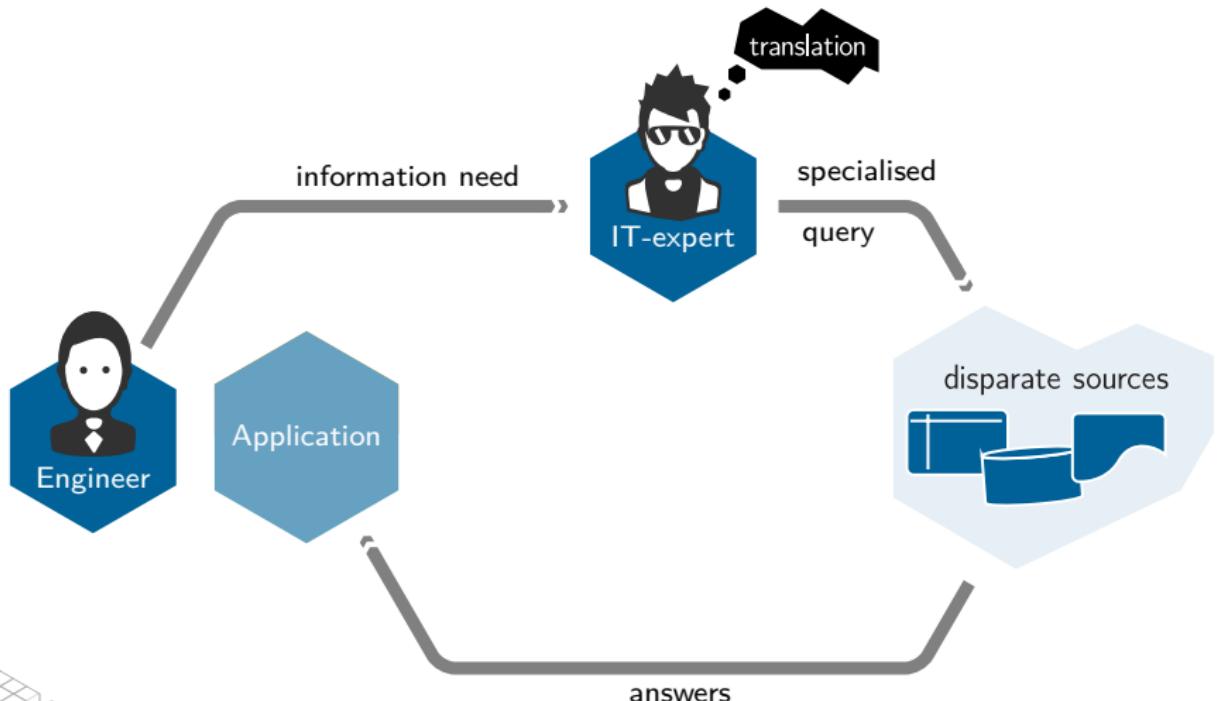
The application **doesn't know about** this data.

They Get Lost !!



# How they Tackle This **Problem**

Optique



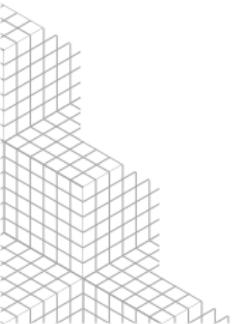
# How they Tackle This **Problem**

Optique



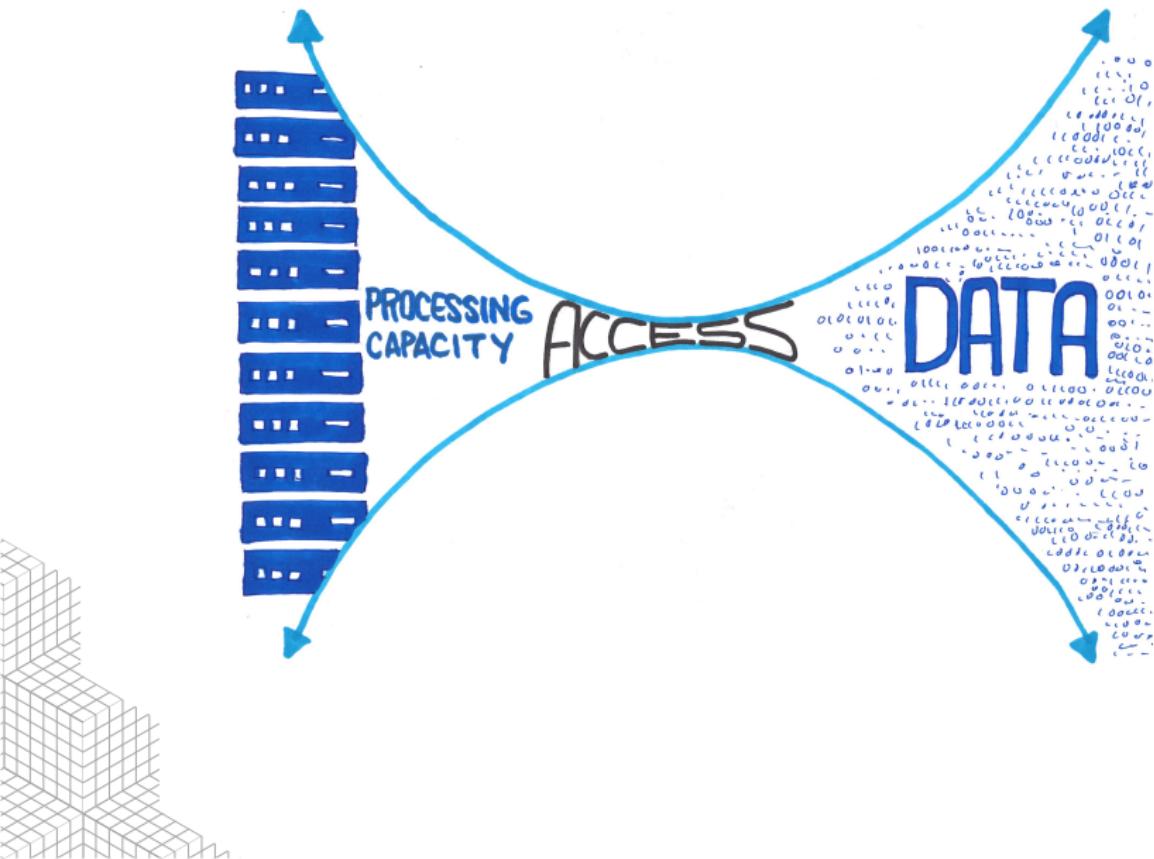
- May take **weeks** to respond.
- Takes several years to master data stores and user needs.

Siemens loses up to **50.000.000** euro per year because of this!!

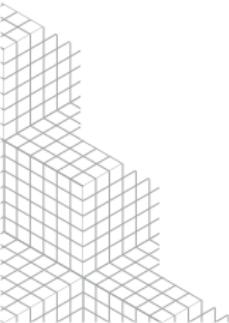


# Data Access **Bottleneck**

Optique

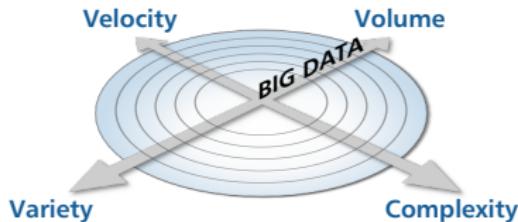


- Provides a semantic **end-to-end connection** between users and data sources;
- Enables users to rapidly formulate **intuitive** queries using familiar vocabularies and conceptualisations.



# How Optique Tackles the Problem

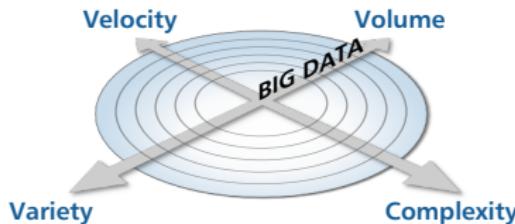
Optique



- (i) Providing scalable big/fast data **infrastructures**.
- (ii) Providing the ability to **cope with variety and complexity** in the data.
- (iii) Providing a good **understanding** of the data.

# How Optique Tackles the Problem

Optique



- (i) Providing scalable big/fast data **infrastructures**.
- (ii) Providing the ability to **cope with variety and complexity** in the data. **(OBDA)**
- (iii) Providing a good **understanding** of the data. **(OBDA)**

- -ontop- is a platform to **query databases** as virtual RDF Graphs **using SPARQL**, and exploiting OWL 2 QL ontologies and mappings
- It is currently being developed in the context of the **Optique** project.
- Development of -ontop- started 4.5 years ago.
- -ontop- is already well established
  - website has +-500 visits per month
  - got 500 new registrations in the past year
- -ontop- is **open-source** and released under Apache license.
- We support all major relational DBs (Oracle, DB2, Postgres, MySQL, etc.)



Introduction: Optique and Ontop

Ontology Based Data Access

The Database:

Ontologies

Mappings

Virtual Graph

Querying

Data Integration

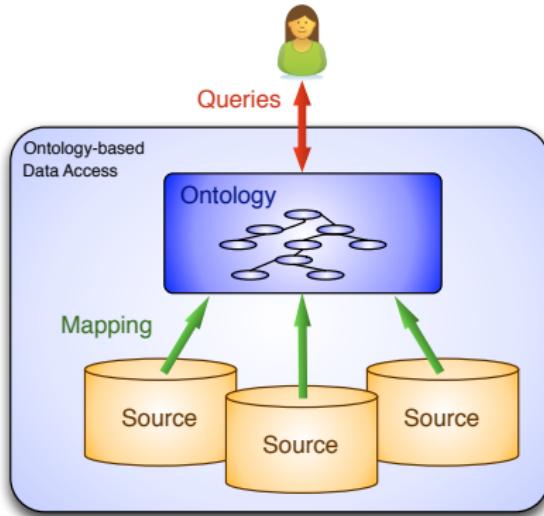
Checking Consistency

Conclusions



# Ontology-based data access (OBDA)

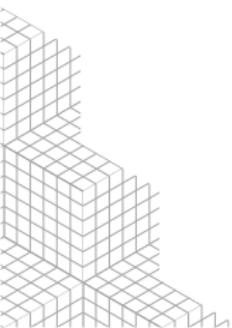
Optique



- **Data source(s)**: are external and independent (possibly multiple and heterogeneous).
- **Ontology**: provides a unified common vocabulary, and a conceptual view of the data.
- **Mappings**: relate each term in the ontology to a set of (SQL) views over (possibly federated) data.

# Simple Life Science Running Example Optique

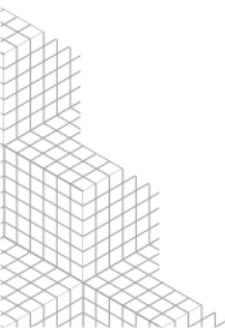
- **Data source(s)**: Hospital Databases with cancer patients (First 1, then 2)
- **Ontology**: A common domain vocabulary defining Patient, Cancer, LungCancer, etc.
- **Mappings**: Relating the vocabulary and the databases.



Before we start you need:

- Java 1.7 (check typing: `java -version`)
- The material online:

[https://www.dropbox.com/sh/316cavgoavjtnu3/  
AAADoau5NuNGq3zX04JJQ6rya?dl=0](https://www.dropbox.com/sh/316cavgoavjtnu3/AAADoau5NuNGq3zX04JJQ6rya?dl=0)



Introduction: Optique and Ontop

Ontology Based Data Access

The Database:

Ontologies

Mappings

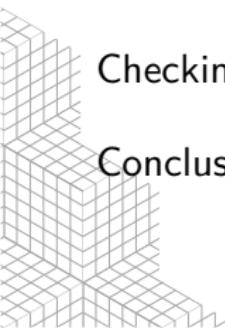
Virtual Graph

Querying

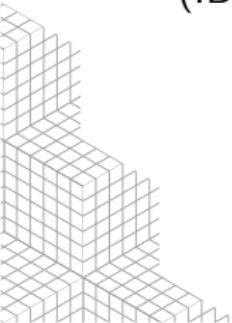
Data Integration

Checking Consistency

Conclusions



- 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)

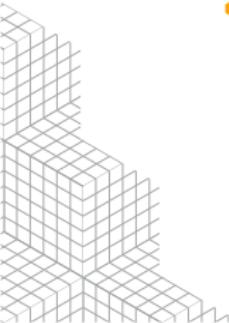
Stage is:

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

# Creating the DB in H2

Optique

- 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:

Optique

The screenshot shows a web browser window with the URL `http://localhost:8082/login.jsp?sessionid=b392a321f25e05f6f4425b41b128d88c`. The page title is "Connexion". It displays a form for configuring a JDBC connection. The configuration is registered as "Generic H2 (Embedded)". The JDBC driver is "org.h2.Driver" and the URL is "jdbc:h2:tcp://localhost/helloworld;DATABASE\_TO\_UPPE". The user name is "sa" and the password field is empty. There are "Connecter" and "Test de connexion" buttons at the bottom.

Configuration enregistrée:	Generic H2 (Embedded)		
Nom de configuration:	Generic H2 (Embedded)	Enregistrer	Supprimer
Pilote JDBC:	org.h2.Driver		
URL JDBC:	jdbc:h2:tcp://localhost/helloworld;DATABASE_TO_UPPE		
Nom d'utilisateur:	sa		
Mot de passe:			

Connecter   Test de connexion

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

# How to access it from the web

Optique

The screenshot shows a Mac OS X desktop environment with a window titled "H2 Icon". The window is a Java-based database console for H2. On the left, there's a sidebar with a tree view of database objects:

- jdbc:h2:tcp://localhost/helloworld
- INFORMATION\_SCHEMA
- Utilisateurs
- H2 1.3.173 (2013-07-28)

The main area has tabs: "Exécuter (Ctrl+Enter)" (selected), "Effacer", and "Instruction SQL:". A placeholder text "Write your queries here!" is visible in the large central text area.

**Tables Info**

**Commandes principales**

- Affiche cette page d'aide
- Affiche l'historique des commandes
- Exécute la commande courante
- Déconnexion de la base de données

**Exemple de script SQL**

Effacer une table si elle existe	DROP TABLE IF EXISTS TEST;
Créer une nouvelle table avec les colonnes ID et NAME	CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
Ajouter un nouvel enregistrement	INSERT INTO TEST VALUES('1', 'Hello');
Ajouter un autre enregistrement	INSERT INTO TEST VALUES('2', 'World');
Requérir une table	SELECT * FROM TEST ORDER BY ID;
Modifier un enregistrement	UPDATE TEST SET NAME='Hi' WHERE ID=1;
Effacer un enregistrement	DELETE FROM TEST WHERE ID=2;
Aide	HELP ...

**Example Queries**

# Creating the table

Optique

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:

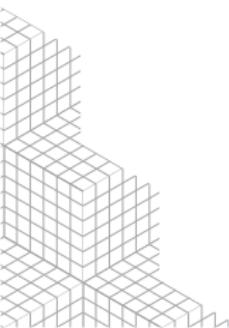
```
INSERT INTO "tbl_patient"
(patientid,name,type,stage)
VALUES
(1,'Mary',false,4),
(2,'John',true,7);
```

# Example **SQL** Query

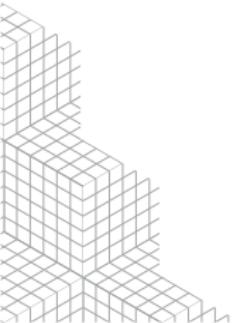
Optique

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



Introduction: Optique and Ontop

Ontology Based Data Access

The Database:

Ontologies

Mappings

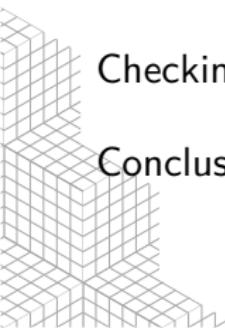
Virtual Graph

Querying

Data Integration

Checking Consistency

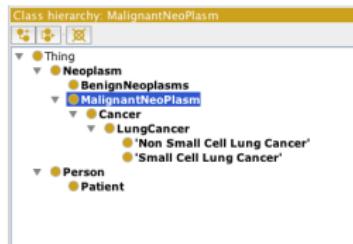
Conclusions



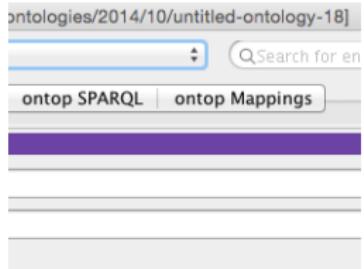
## Definition

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



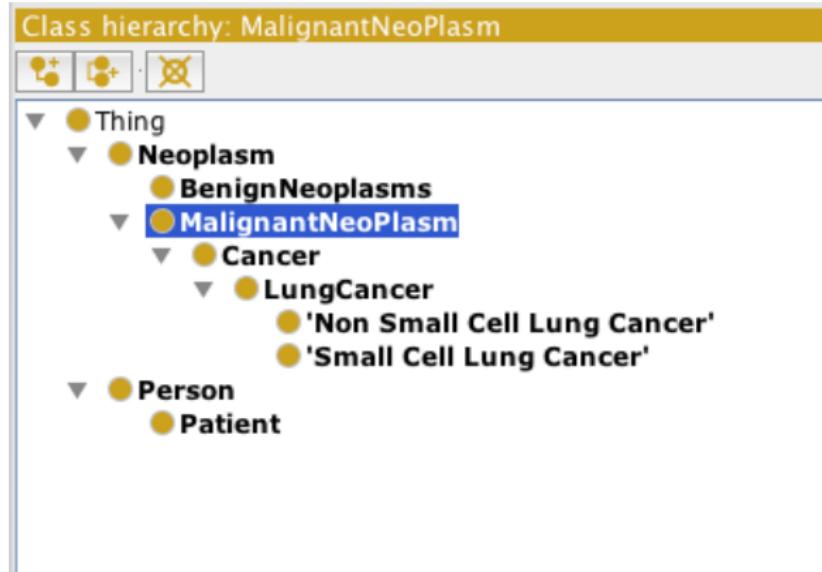
- Go to the protégé-ontop folder from your material. This is a Protégé 4.3 package that includes the ontop plugin
- Run Protégé from the console using the run.bat or run.sh scripts. That is, execute:
- cd Protege\_4.3/; run.sh



# The ontology: Creating Concepts and Properties

Optique

Add the concept: Patient.



(See PatientOnto.owl)

# The ontology: Creating Concepts and Properties

Optique

Add these object properties:

Object property hierarchy: hasStage

Object property hierarchy: hasStage

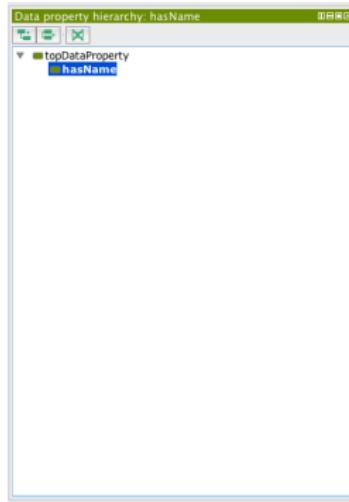
topObjectProperty

- hasNeoplasm
- hasStage

# The ontology: Creating Concepts and Properties

Optique

Add these data properties:



(See PatientOnto.owl)

Introduction: Optique and Ontop

Ontology Based Data Access

The Database:

Ontologies

Mappings

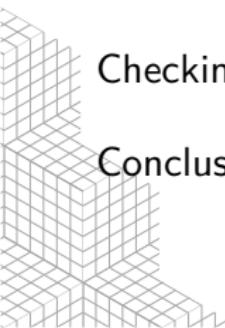
Virtual Graph

Querying

Data Integration

Checking Consistency

Conclusions



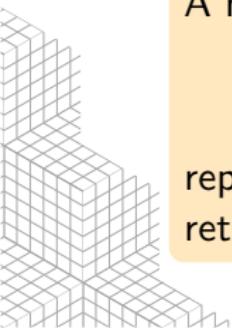
- We have the vocabulary, the database, now we need to link those two.
- Mappings define triples (subject, property, object) out of SQL queries.
- These triples are accessible during query time (the on-the-fly approach) or can be imported into the OWL ontology (the ETL approach)

## Definition (Intuition)

A mapping has the following form:

*TripleTemplate* ← SQL Query to build the triples

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

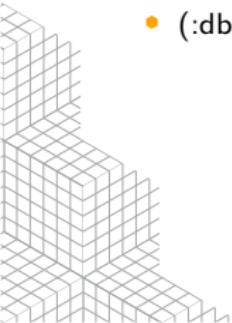


# The Mappings

Optique

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

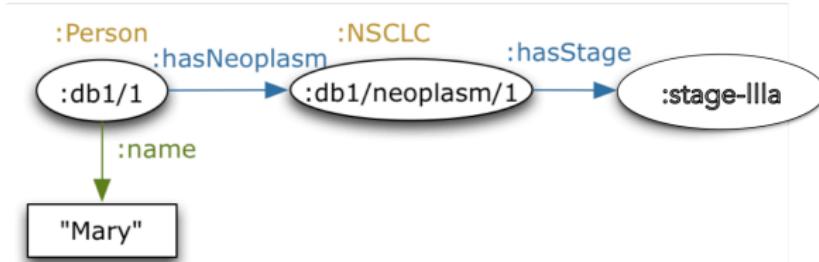


# The Mappings

Optique

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



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

Datasource editor:

OBDA Model information

Number of sources: 1

Connection parameters

Datasource Name: patientDB

Connection URL: jdbc:h2:tcp://localhost/helloworld

Database Username: sa

Database Password:

Driver class: org.h2.Driver

 Test Connection      Connection is OK

Datasource Manager | Mapping Manager | Mapping Assistant - BETA

# The Mappings

Optique

- 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

Optique

- 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
```

Introduction: Optique and Ontop

Ontology Based Data Access

The Database:

Ontologies

Mappings

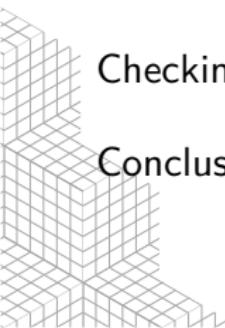
Virtual Graph

Querying

Data Integration

Checking Consistency

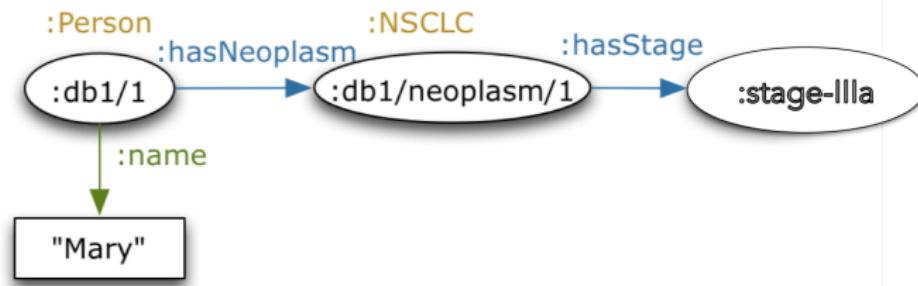
Conclusions



# Virtual Graph

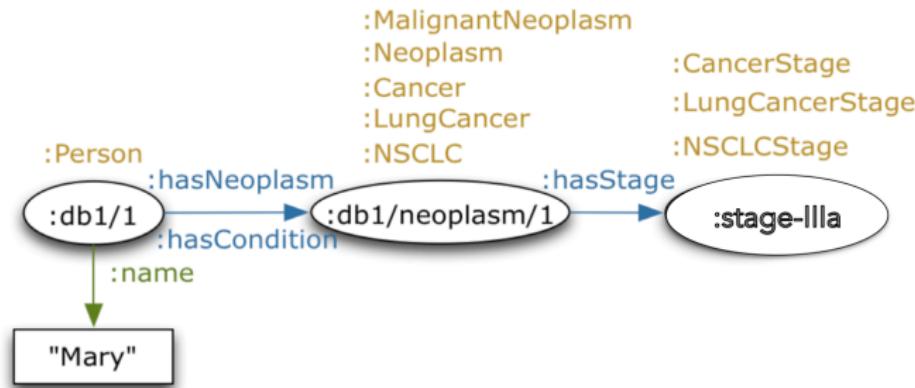
Optique

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.

Introduction: Optique and Ontop

Ontology Based Data Access

The Database:

Ontologies

Mappings

Virtual Graph

Querying

Data Integration

Checking Consistency

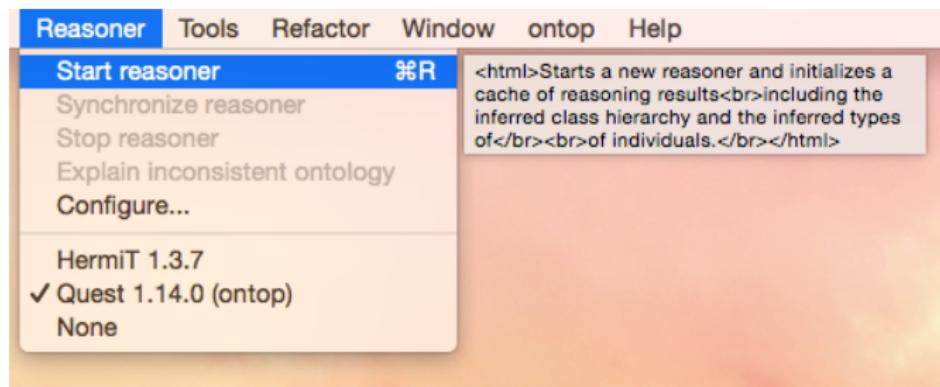
Conclusions



# On-the-fly access to the DB

Optique

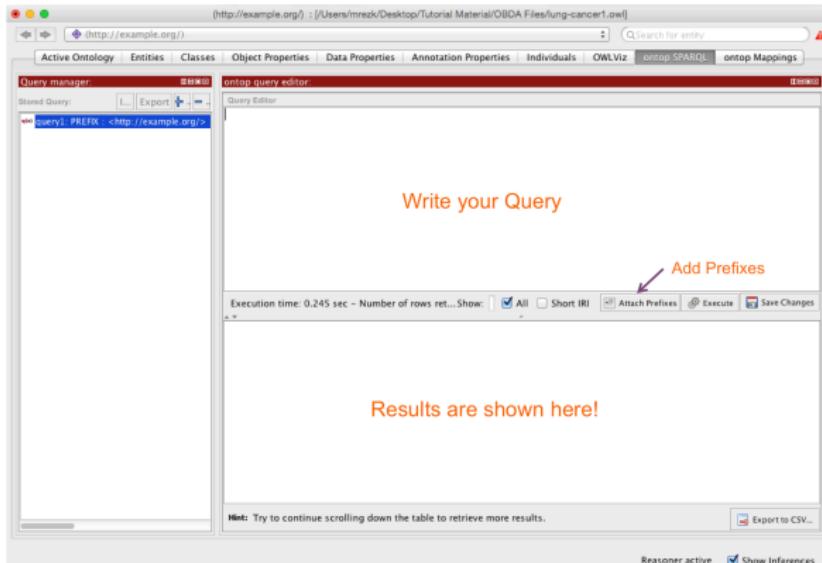
- Recall our information need: Give me the id and the name of the patients with a tumor at stage IIIa.
- Enable Ontop in the “Reasoner” menu



# On-the-fly access to the DB

Optique

In the ontop SPARQL tab add all the prefixes



# On-the-fly access to the DB

Optique

- Write the SPARQL Query

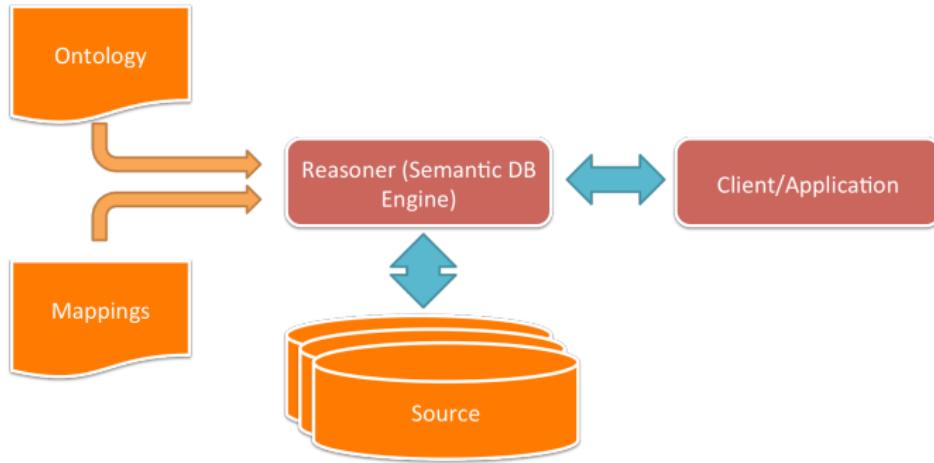
```
SELECT ?p ?name WHERE
{ ?p rdf:type :Patient .
?p :hasName ?name .
?p :hasNeoplasm ?tumor .
?tumor :hasStage :stage-IIIa .}
```

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



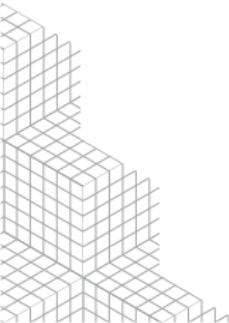
# How we do Inference...

Optique



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

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



Introduction: Optique and Ontop

Ontology Based Data Access

The Database:

Ontologies

Mappings

Virtual Graph

Querying

**Data Integration**

Checking Consistency

Conclusions

# 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

Edit Mapping

Mapping ID: map-patient

Target (Triples Template):  
:db2/{PATIENTID} a :Patient ; :name {NAME} .

Source (SQL Query):  
select \* from T\_Name

Test SQL Query

Update    Cancel

# New Mappings

Optique

Edit Mapping

Mapping ID: nsclc

Target (Triples Template):  
`:db2/neoplasm/{PATIENTID} a :NSCLC .`

Source (SQL Query):  
`select * from T_NSCLC`

Test SQL Query

Update Cancel

Edit Mapping

Mapping ID: map-patient

Target (Triples Template):  
`:db2/{PATIENTID} a Patient ; name (NAME) .`

Source (SQL Query):  
`select * from T_Name`

Test SQL Query

Update Cancel

# New Mappings

Optique

The image displays two separate windows titled "Edit Mapping".

**Left Window (Mapping ID: map-patient):**

- Target (Triples Template):** `:db2/(PATIENTID) a :Patient ; :name [NAME] .`
- Source (SQL Query):** `select * from T_Name`
- Buttons:** "Test SQL Query", "Update" (green checkmark), and "Cancel" (red X).

**Right Window (Mapping ID: nsclc):**

- Target (Triples Template):** `:db2/neoplasm/(PATIENTID) a :NSCLC .`
- Source (SQL Query):** `select * from T_NSCLC`
- Buttons:** "Test SQL Query", "Update" (green checkmark), and "Cancel" (red X).

- 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

Introduction: Optique and Ontop

Ontology Based Data Access

The Database:

Ontologies

Mappings

Virtual Graph

Querying

Data Integration

Checking Consistency

Conclusions

- 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

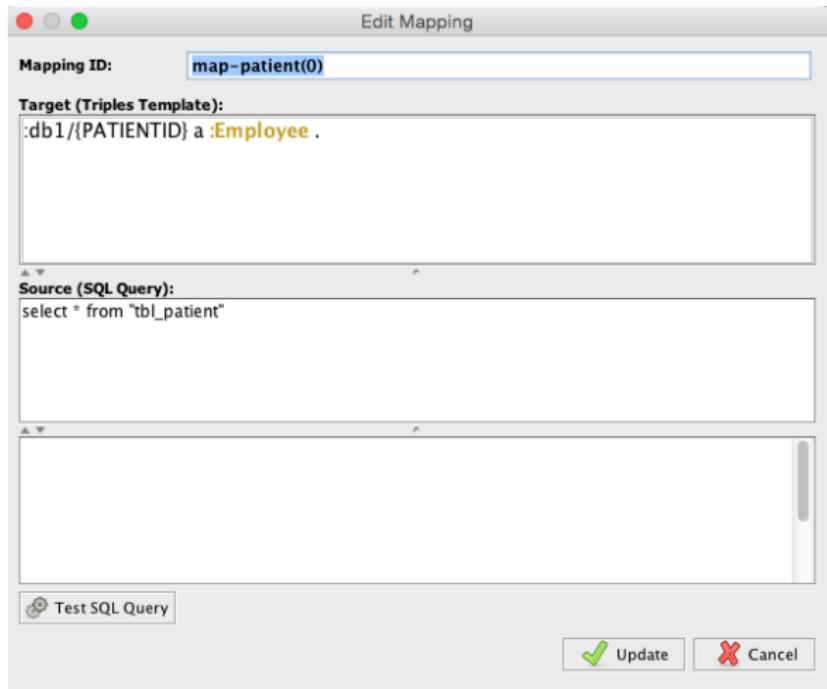
Optique

The screenshot shows the Optique ontology editor interface with the following panels:

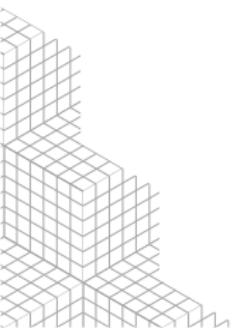
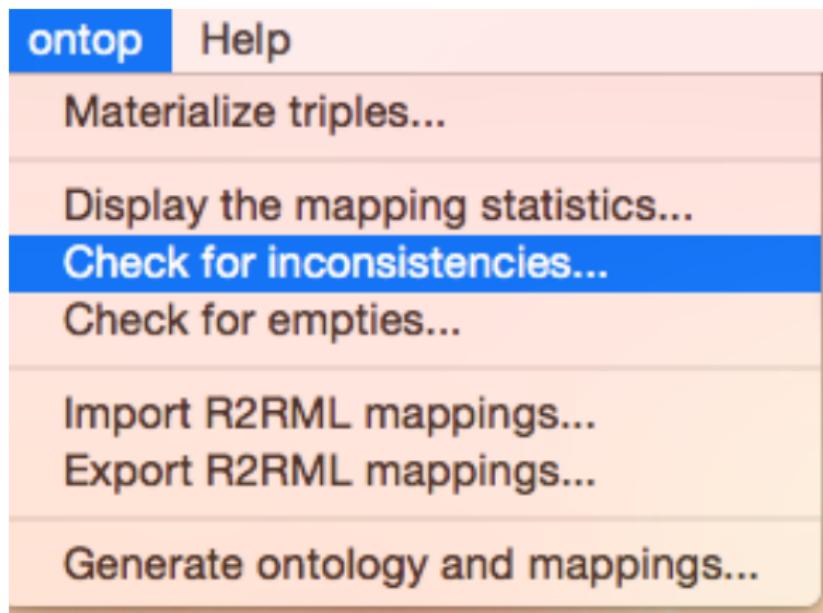
- Class hierarchy: Employee**: Shows a tree structure with **Thing** at the root, expanded to show **Employee** and **Patient**.
- Annotations: Employee**: An empty panel for managing annotations.
- Description: Employee**: A detailed view of the **Employee** class:
  - Equivalent To**: No entries.
  - SubClass Of**: No entries.
  - SubClass Of (Anonymous Ancestor)**: No entries.
  - Members**: No entries.
  - Target for Key**: No entries.
  - Disjoint With**: Shows **Patient** as a disjoint member.
  - Disjoint Union Of**: No entries.
- Object property hierarchy** and **Data property hierarchy**: Both show a single entry: **topObjectProperty**.

At the bottom, a status bar indicates: "Reasoner state out of sync with active ontology" and "Show Inferences".

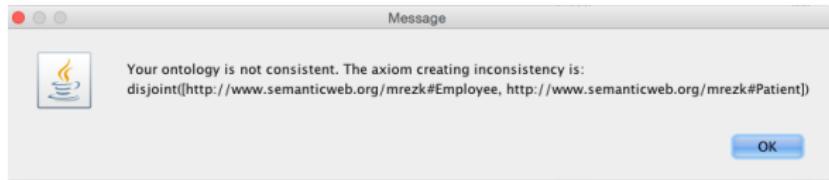
# Consistency: Building a wrong mapping Optique



# Consistency: Checking Inconsistency Optique



# Consistency: Finding out the Problem Optique



Introduction: Optique and Ontop

Ontology Based Data Access

The Database:

Ontologies

Mappings

Virtual Graph

Querying

Data Integration

Checking Consistency

Conclusions

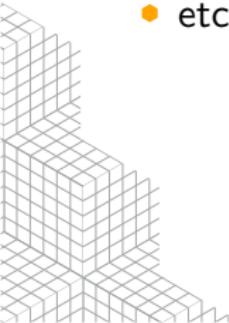
- Ontologies give you a common vocabulary to formulate the **queries**, and mappings to find the **answers**.
- Ontologies and Semantic Technology can help to handle the problem of **accessing Big Data**
  - **Diversity:**
    - Using ontologies describing particular domains allows to **hide** the storage 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...

Optique

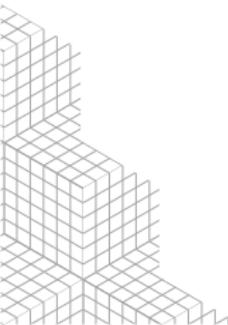
- Semantic Query Optimisation
- SWRL and Recursion
- Performance Evaluation
- Aggregates and bag semantics
- Give out about Database Engines
- Tons of theory
- etc. etc. etc...



Thanks!!!

THANKS!!!

<http://ontop.inf.unibz.it>  
[www.optique-project.eu](http://www.optique-project.eu)

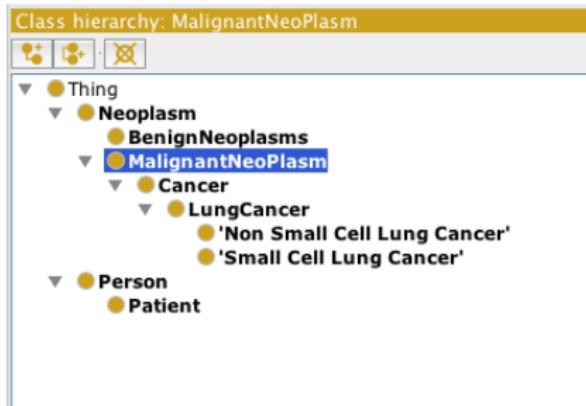


# Extra: Where Reasoning takes Place

Optique

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

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



# Extra: Where Reasoning takes Place

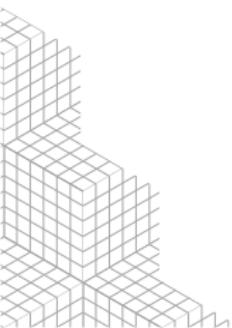
Optique

- 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). } }
```



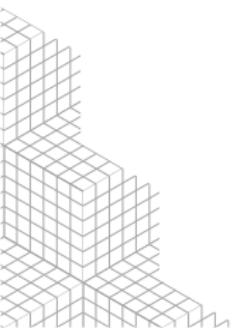
# Extra: Where Reasoning takes Place Optique

- 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). } }
```



# Extra: 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 :Neoplasms}. }
```

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

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

