

INF3580/4580 – Semantic Technologies – Spring 2018

Lecture 14: Introduction to Ontology-based Data Access (OBDA)

Ernesto Jimenez Ruiz

15th May 2018



DEPARTMENT OF
INFORMATICS



UNIVERSITY OF
OSLO

Today's Plan

- 1 Exposing data as RDF
- 2 Data Access in Statioil: limitations and solutions
- 3 OBDA Ingredients
 - Overview
 - Ontology
 - Mappings
 - Query rewriting
 - Bootstrapping
 - Visual Query Formulation
 - Optique

Outline

- 1 Exposing data as RDF
- 2 Data Access in Statioil: limitations and solutions
- 3 OBDA Ingredients
 - Overview
 - Ontology
 - Mappings
 - Query rewriting
 - Bootstrapping
 - Visual Query Formulation
 - Optique

RDF

- Why URIs?
 - URIs naturally have a “global” scope, unique throughout the web.
 - URIs are also addresses.
 - “*A web of data.*”

RDF

- Why URIs?
 - URIs naturally have a “global” scope, unique throughout the web.
 - URLs are also addresses.
 - “*A web of data.*”
- Why triples?
 - Any information format can be transformed to triples.
 - Relationships are made explicit and are elements in their own right

RDF on the web: Where is it?

- In files:
 - In some serialisation format: XML/RDF, Turtle, ...
 - Typically small RDF graphs, i.e., max. a few 100 triples, e.g.,
 - Vocabularies: <http://xmlns.com/foaf/spec/index.rdf>.
 - Tiny datasets: <http://folk.uio.no/martingi/foaf.rdf>.

RDF on the web: Where is it?

- In files:
 - In some serialisation format: XML/RDF, Turtle, ...
 - Typically small RDF graphs, i.e., max. a few 100 triples, e.g.,
 - Vocabularies: <http://xmlns.com/foaf/spec/index.rdf>.
 - Tiny datasets: <http://folk.uio.no/martingi/foaf.rdf>.
- “Behind” *SPARQL endpoints*:
 - Data kept in a *triple store*, i.e., a database of triples.
 - RDF is served from endpoints as results of *SPARQL queries*.
 - Exposes data (in different ways)
 - with endpoint frontends, e.g., <http://dbpedia.org/resource/Norway>, or
 - by direct SPARQL query: <http://dbpedia.org/sparql>.

RDF on the web: Where is it?

- “Behind” *OBDA repositories*:
 - OBDA: Ontology-based Data Access
 - Data kept in a traditional *relational database*
 - Access is transparent via SPARQL queries
 - SPARQL queries are “internally” transformed to SQL queries
 - An RDF representation of the relational database is “virtualized” to answer the SPARQL query.

Exposing data as RDF

- **Virtual exposure of data (OBDA)**
 - ✓ End-users' friendly access to “unfriendly” relational data
 - ✓ Pay as you go data integration
 - ✗ Requires an ontology in OWL 2 QL
 - ✗ Data remains in old-fashioned databases

Exposing data as RDF

- **Virtual exposure of data (OBDA)**

- ✓ End-users' friendly access to “unfriendly” relational data
- ✓ Pay as you go data integration
- ✗ Requires an ontology in OWL 2 QL
- ✗ Data remains in old-fashioned databases

- **Data Export**

- ✓ Easy to exchange data (over the Web)
- ✓ Ontology not limited to OWL 2 QL
- ✗ Data replication
- ✗ Due to size or privacy it may not be possible to export the data

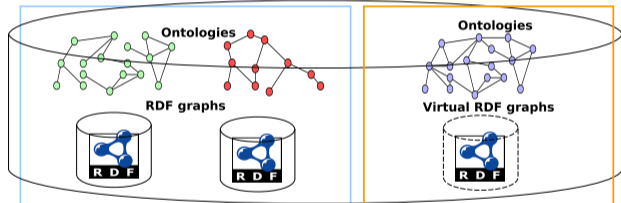
Exposing data as RDF

Applications

- Complex data access
- Analogues search
- Analytics
- Data visualization



Knowledge Graphs



Resources

Exposed as Knowledge Graphs



Ontology Based Data Access



Outline

- 1 Exposing data as RDF
- 2 Data Access in Statioil: limitations and solutions
- 3 OBDA Ingredients
 - Overview
 - Ontology
 - Mappings
 - Query rewriting
 - Bootstrapping
 - Visual Query Formulation
 - Optique

EU project Optique

- EU Project from 2012-2016
- Aimed at facilitating **scalable end-user access to big data** in the oil and gas industry.
- Advocated for an **OBDA approach**
 - ontology provides a virtual access to the data
 - mappings connect the ontology with the data source.

EU project Optique

- EU Project from 2012-2016
- Aimed at facilitating **scalable end-user access to big data** in the oil and gas industry.
- Advocated for an **OBDA approach**
 - ontology provides a virtual access to the data
 - mappings connect the ontology with the data source.
- Focused around two demanding use cases provided by the industry partners **Siemens** and **Statoil**
- Currently takes 30-70% of engineers' time (e.g., more than 250 MNOK annually).

Limitations

Simple case:



Application

predefined queries

uniform sources



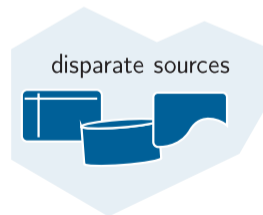
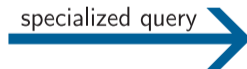
Limitations

All norwegian wellbores of [type] nearby [place]
having a permeability near [value]. [...]
Attributes: completion date, depth, etc.

Complex case:



translation



Takes 4 days in average

A typical query at Statoil

Anonymized extract

```

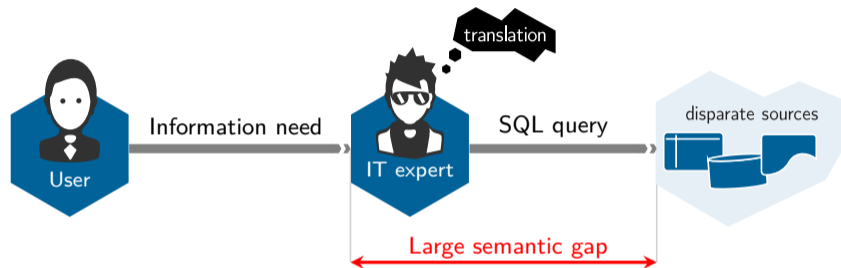
SELECT [...]
FROM
db_name.table1 table1,
db_name.table2 table2a,
db_name.table2 table2b,
db_name.table3 table3a,
db_name.table3 table3b,
db_name.table3 table3c,
db_name.table3 table3d,
db_name.table4 table4a,
db_name.table4 table4b,
db_name.table4 table4c,
db_name.table4 table4d,
db_name.table4 table4e,
db_name.table4 table4f,
db_name.table5 table5a,
db_name.table5 table5b,
db_name.table6 table6a,
db_name.table6 table6b,
db_name.table7 table7a,
db_name.table7 table7b,
db_name.table8 table8,
db_name.table9 table9,
db_name.table10 table10a,
db_name.table10 table10b,
db_name.table10 table10c,
db_name.table11 table11,
db_name.table12 table12,
db_name.table13 table13,
db_name.table14 table14,
db_name.table15 table15,
db_name.table16 table16
WHERE [...]

table2a.attr1='keyword' AND
table3a.attr2=table10c.attr1 AND
table3a.attr6=table6a.attr3 AND
table3a.attr9='keyword' AND
table4a.attr10 IN ('keyword') AND
table4a.attr1 IN ('keyword') AND
table5a.kinds=table4a.attr13 AND
table5b.kinds=table4c.attr74 AND
table5b.name='keyword' AND
(table6a.attr19=table10c.attr17 OR
(table6a.attr2 IS NULL AND
table10c.attr4 IS NULL)) AND
table6a.attr14=table5b.attr14 AND
table6a.attr2='keyword' AND
(table6b.attr14=table10c.attr8 OR
(table6b.attr4 IS NULL AND
table10c.attr7 IS NULL)) AND
table6b.attr19=table5a.attr55 AND
table6b.attr2='keyword' AND
table7a.attr19=table2b.attr19 AND
table7a.attr17=table15.attr19 AND
table4b.attr11='keyword' AND
table8.attr19=table7a.attr80 AND
table8.attr19=table13.attr20 AND
table8.attr4='keyword' AND
table9.attr10=table16.attr11 AND
table3b.attr19=table10c.attr18 AND
table3b.attr22=table12.attr63 AND
table3b.attr66='keyword' AND
table10a.attr54=table7a.attr8 AND
table10a.attr70=table10c.attr10 AND
table10a.attr16=table4d.attr11 AND
table4c.attr99='keyword' AND
table4c.attr1='keyword' AND

table11.attr10=table5a.attr10 AND
table11.attr40='keyword' AND
table11.attr50='keyword' AND
table2b.attr1=table1.attr8 AND
table2b.attr9 IN ('keyword') AND
table2b.attr2 LIKE 'keyword'%' AND
table12.attr9 IN ('keyword') AND
table7b.attr1=table2a.attr10 AND
table3c.attr13=table10c.attr1 AND
table3c.attr10=table6b.attr20 AND
table3c.attr13='keyword' AND
table10b.attr16=table10a.attr7 AND
table10b.attr11=table7b.attr8 AND
table10b.attr13=table4b.attr89 AND
table13.attr1=table2b.attr10 AND
table13.attr20='keyword' AND
table13.attr15='keyword' AND
table3d.attr49=table12.attr18 AND
table3d.attr18=table10c.attr11 AND
table3d.attr14='keyword' AND
table4d.attr17 IN ('keyword') AND
table4d.attr19 IN ('keyword') AND
table16.attr28=table11.attr56 AND
table16.attr16=table10b.attr78 AND
table16.attr5=table14.attr56 AND
table4e.attr34 IN ('keyword') AND
table4e.attr48 IN ('keyword') AND
table4f.attr89=table5b.attr7 AND
table4f.attr45 IN ('keyword') AND
table4f.attr1='keyword' AND
table10c.attr2=table4e.attr19 AND
(table10c.attr78=table12.attr56 OR
(table10c.attr55 IS NULL AND
table12.attr17 IS NULL))

```

Limitations



Querying over RDB requires a lot of knowledge about:

- Magic numbers (e.g., $1 \rightarrow$ full professor)
- Cardinalities and normal forms
- Relevant and closely-related information spread over many tables

High-level translation solution

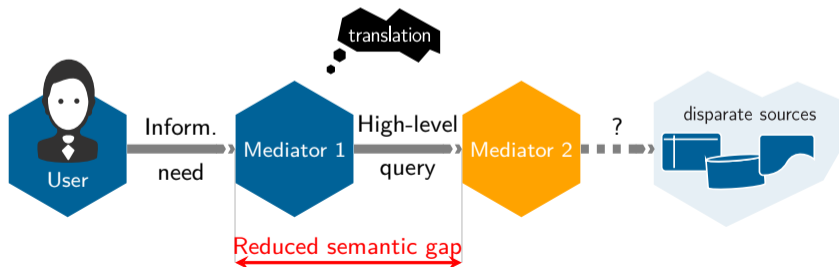
General approach: two steps

- 1 Translate the information needs into a **high-level (formal) query**
 - 1 *Mediator 1* could be a user, an IT expert or a GUI
 - 2 Make such a translation easy (*Ideally: IT expertise not required*)
- 2 Answer the high-level query **automatically** using *Mediator 2*

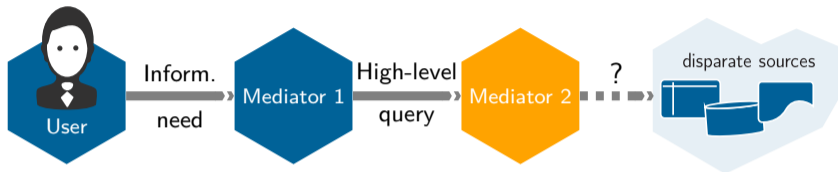
High-level translation solution

General approach: two steps

- 1 Translate the information needs into a **high-level (formal) query**
 - 1 *Mediator 1* could be a user, an IT expert or a GUI
 - 2 Make such a translation easy (*Ideally: IT expertise not required*)
- 2 Answer the high-level query **automatically** using *Mediator 2*



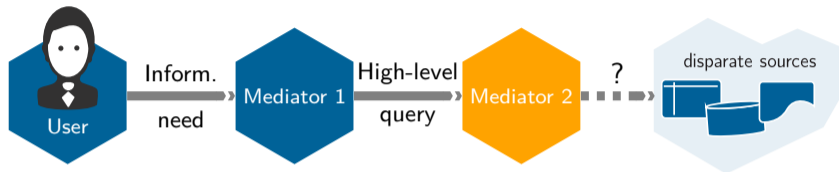
Two orthogonal choices to be made



Choice 1: Generating a new representation of the data

- 1 Extract Transform Load (ETL) process
- 2 Virtual views

Two orthogonal choices to be made



Choice 1: Generating a new representation of the data

- 1 Extract Transform Load (ETL) process
- 2 Virtual views

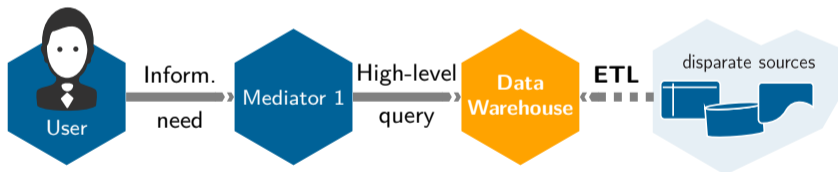
Choice 2: Which data format for the new representation

- 1 New relational schema
- 2 JSON (or XML) documents
- 3 Resource Description Framework (RDF)

Generating a new representation of the data

1. Extract Transform Load (ETL)

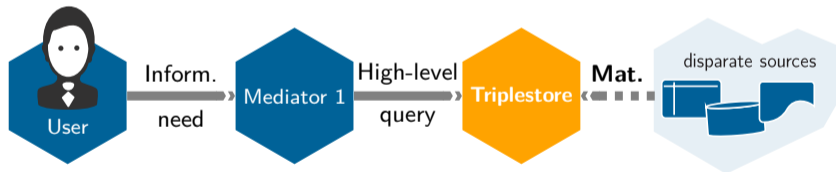
E.g., relational data warehouse



Generating a new representation of the data

1. Extract Transform Load (ETL) / Materialization

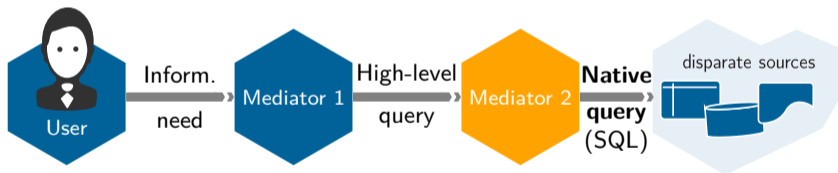
E.g., relational data warehouse, triplestore (RDF)



Generating a new representation of the data

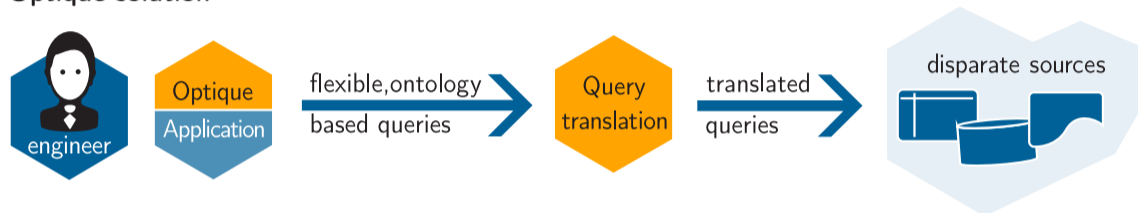
2. Virtual views

E.g., virtual databases (Teiid, Apache Drill, Exareme), **OBDA with Optique**



Optique solution: Ontology-Based Data Access (OBDA)

Optique solution



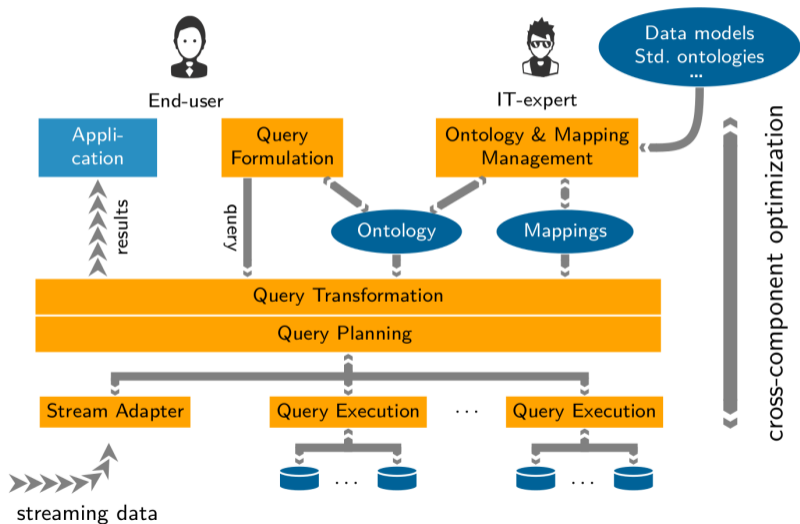
Choice 1: Generating a new representation of the data

- 1 Virtual views

Choice 2: Which data format for the virtual view

- 1 Resource Description Framework (RDF)

Optique architecture



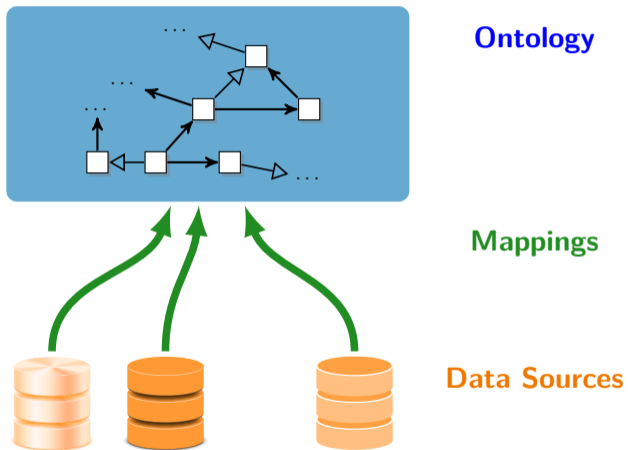
Outline

- 1 Exposing data as RDF
- 2 Data Access in Statioil: limitations and solutions
- 3 **OBDA Ingredients**
 - Overview
 - Ontology
 - Mappings
 - Query rewriting
 - Bootstrapping
 - Visual Query Formulation
 - Optique

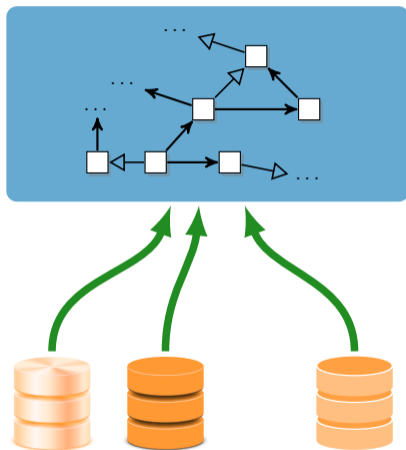
Outline

- 1 Exposing data as RDF
- 2 Data Access in Statioil: limitations and solutions
- 3 OBDA Ingredients**
 - **Overview**
 - Ontology
 - Mappings
 - Query rewriting
 - Bootstrapping
 - Visual Query Formulation
 - Optique

OBDA framework



OBDA framework

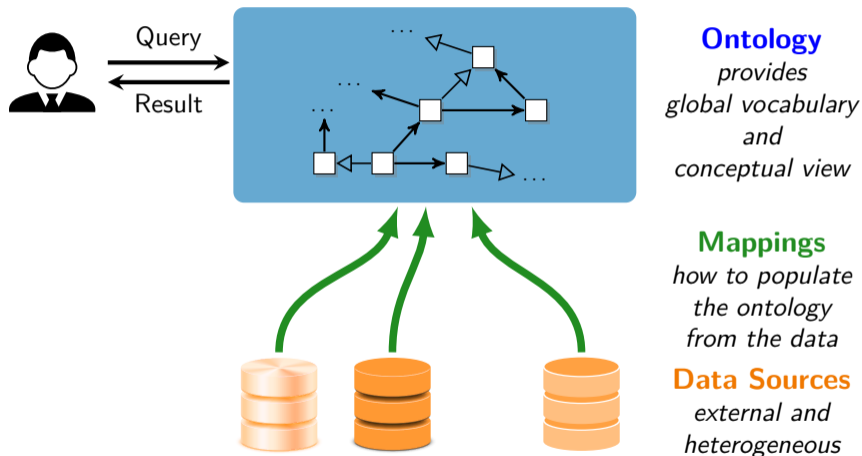


Ontology
*provides
global vocabulary
and
conceptual view*

Mappings
*how to populate
the ontology
from the data*



Data Sources
*external and
heterogeneous*

OBDA framework



OBDA framework

Logical transparency in accessing data:

-  does not know where and how the **data** is stored.
-  can only see a **conceptual view** of the **data**.

OBDA Ingredients

- Relies on...
 - **ontology** to provide a virtual access to the data
 - **mappings** to connect the ontology with the data

- Required infrastructure
 - **Query formulation system** to express the information needs in SPARQL (Mediator 1)
 - **Query transformation/rewriting** system to convert from SPARQL to (native) SQL (Mediator 2)
 - **Ontology and mappings bootstrapper**

Outline

- 1 Exposing data as RDF
- 2 Data Access in Statioil: limitations and solutions
- 3 OBDA Ingredients**
 - Overview
 - Ontology**
 - Mappings
 - Query rewriting
 - Bootstrapping
 - Visual Query Formulation
 - Optique

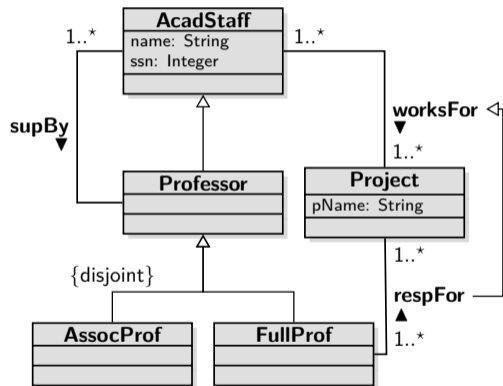
The Ontology: OWL 2 QL profile

- OWL 2 QL is one of the three standard profiles of OWL 2.
- Is considered a lightweight ontology language:
 - controlled expressive power
 - efficient inference

The Ontology: OWL 2 QL profile

- OWL 2 QL is one of the three standard profiles of OWL 2.
- Is considered a lightweight ontology language:
 - controlled expressive power
 - efficient inference
- Optimized for accessing large amounts of data (i.e., for data complexity):
 - *First-order rewritability* of query answering: queries over the ontology can be rewritten into SQL queries over the underlying relational database.
 - Consistency checking is also first-order rewritable.
- The ontology data (ABox) in an OBDA setting is (usually) implicitly defined through the database and mappings.

Capturing UML class diagrams/ER schemas in OWL 2 QL



Professor	\sqsubseteq	AcadStaff
AssocProf	\sqsubseteq	Professor
FullProf	\sqsubseteq	Professor
AssocProf	\sqsubseteq	\neg FullProf
AcadStaff	\sqsubseteq	\exists ssn
\exists ssn	\sqsubseteq	AcadStaff
\exists ssn ⁻	\sqsubseteq	Integer
\exists worksFor	\sqsubseteq	AcadStaff
\exists worksFor ⁻	\sqsubseteq	Project
AcadStaff	\sqsubseteq	\exists worksFor
Project	\sqsubseteq	\exists worksFor ⁻
respFor	\sqsubseteq	worksFor

Outline

- 1 Exposing data as RDF
- 2 Data Access in Statioil: limitations and solutions
- 3 OBDA Ingredients**
 - Overview
 - Ontology
 - Mappings**
 - Query rewriting
 - Bootstrapping
 - Visual Query Formulation
 - Optique

OBDA Mappings

Global-As-View (GAV) mapping assertion $\varphi \rightsquigarrow \psi$

- φ : FO query
- ψ : atom
- Open-World Assumption (by default)

Class instance (:Student)

Source	$q(s) \leftarrow \text{uni1-student}(s, f, l)$ <pre>SELECT s_id FROM uni1.student</pre>
Target	$\text{Student}(\text{URI}_1(s))$ <pre>ex:uni1/student/{s_id} a :Student .</pre>

OBDA Mappings

Global-As-View (GAV) mapping assertion $\varphi \rightsquigarrow \psi$

- φ : FO query (over DB predicates)
- ψ : atom (over an RDF predicate)
- Open-World Assumption (by default)

Class instance (:Student)

Source	$q(s) \leftarrow \text{uni1-student}(s, f, l)$ <pre>SELECT s_id FROM uni1.student</pre>
Target	$\text{Student}(\text{URI}_1(s))$ <pre>ex:uni1/student/{s_id} a :Student .</pre>

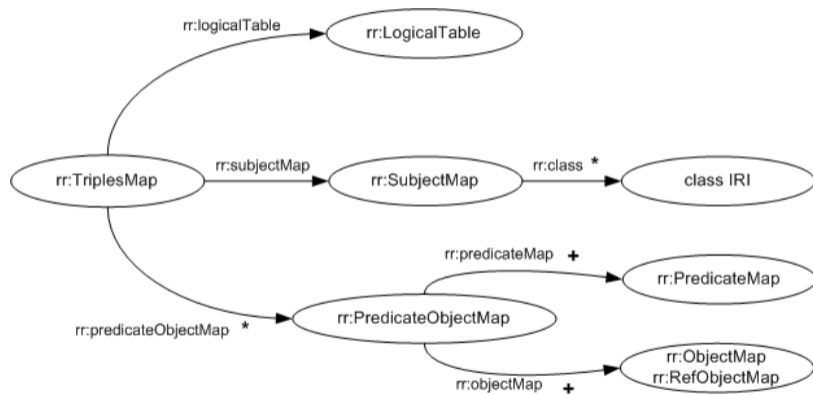
OBDA Mappings: R2RML

- R2RML is a W3C recommended RDB-to-RDF mapping language
 - <https://www.w3.org/TR/r2rml/>
- Generates RDF triples from a relational database based on specific mappings
- The mappings are specified in Turtle syntax

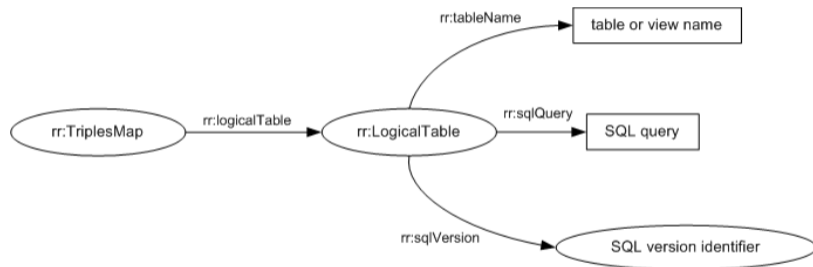
OBDA Mappings: R2RML

- R2RML is a W3C recommended RDB-to-RDF mapping language
 - <https://www.w3.org/TR/r2rml/>
- Generates RDF triples from a relational database based on specific mappings
- The mappings are specified in Turtle syntax
- The R2RML mapping is an RDF graph consisting of several `rr:TriplesMaps`
 - how to map a logical table in the input relational database into RDF

OBDA Mappings: R2RML



OBDA Mappings: R2RML



OBDA Mappings: R2RML example

Triples map to populate Student class"

```
<TriplesMap1>      a rr:TriplesMapClass;
  rr:logicalTable [rr:SQLQuery "Select s_id, name, from STUDENT"];
  rr:subjectMap
  [
    rr:template "http://example.com/uni1/student/{s_id}";
    rr:class ex:Student;
  ] ;
  rr:predicateObjectMap
  [
    rr:predicate foaf:name;
    rr:objectMap [ rr:column "name" ]
  ] .
```

OBDA Mappings: R2RML example

Table STUDENT:

s_id	name
1	Ernesto
2	Martin
3	Leif

OBDA Mappings: R2RML example

Table STUDENT:

s_id	name
1	Ernesto
2	Martin
3	Leif

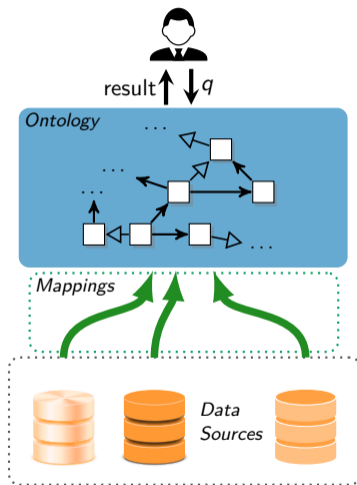
Triples:

```
http://example.com/uni1/student/1    foaf:name    Ernesto
http://example.com/uni1/student/2    foaf:name    Martin
http://example.com/uni1/student/3    foaf:name    Leif
```

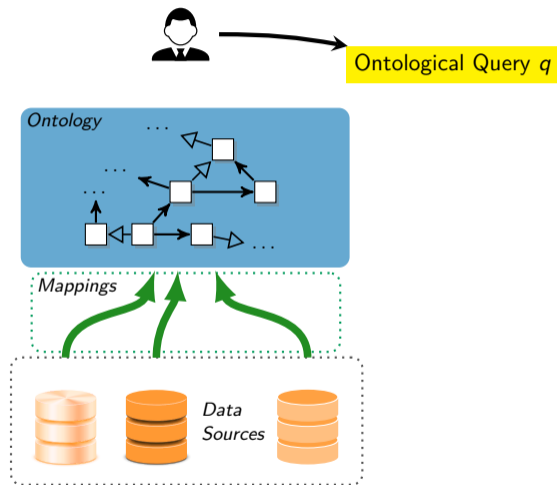

Outline

- 1 Exposing data as RDF
- 2 Data Access in Statioil: limitations and solutions
- 3 OBDA Ingredients**
 - Overview
 - Ontology
 - Mappings
 - Query rewriting**
 - Bootstrapping
 - Visual Query Formulation
 - Optique

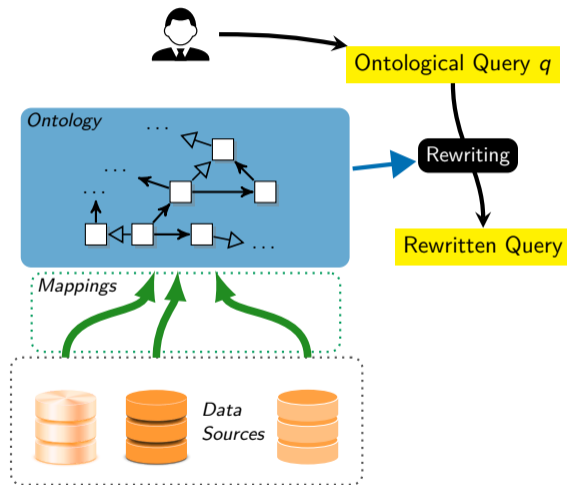
Query answering by rewriting



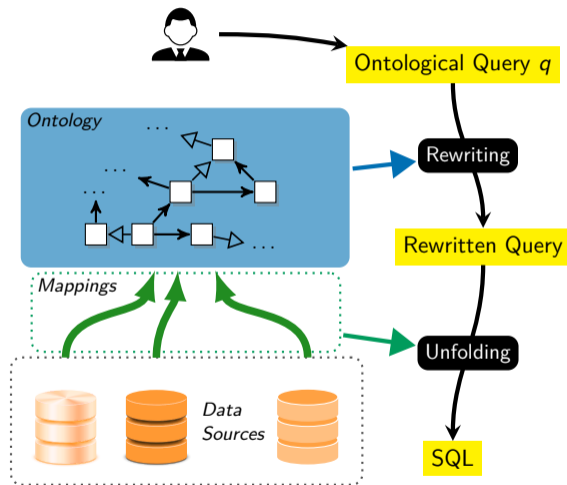
Query answering by rewriting



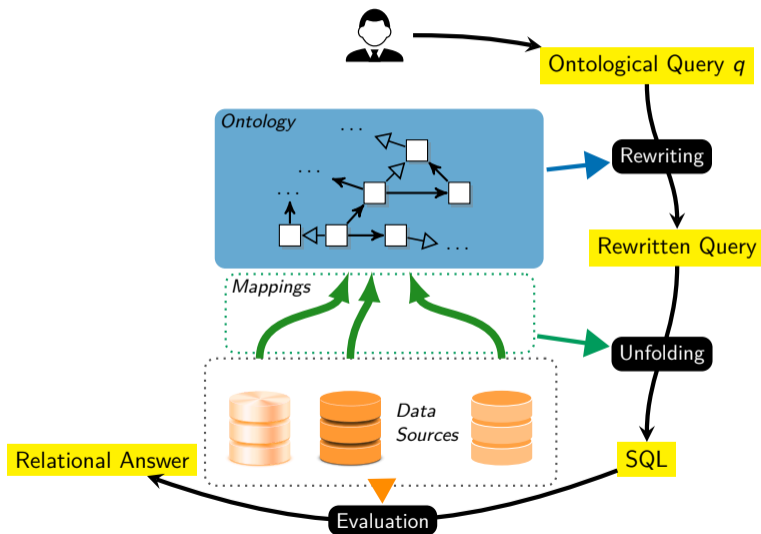
Query answering by rewriting



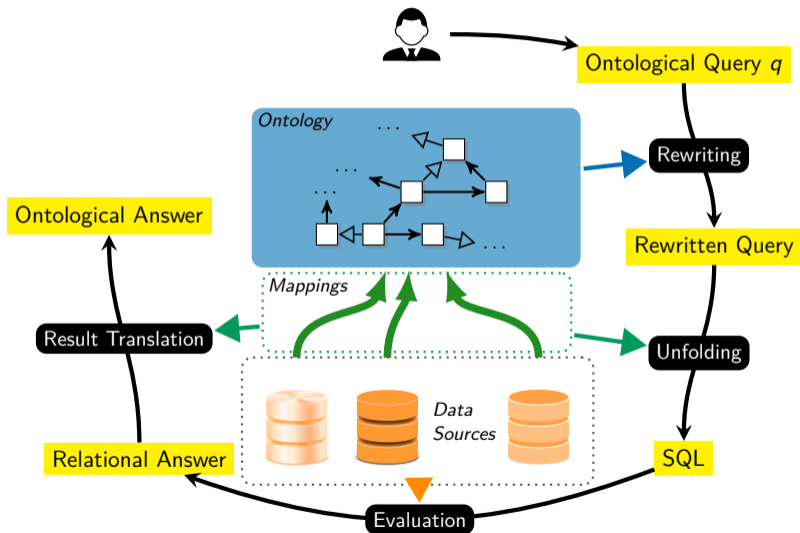
Query answering by rewriting



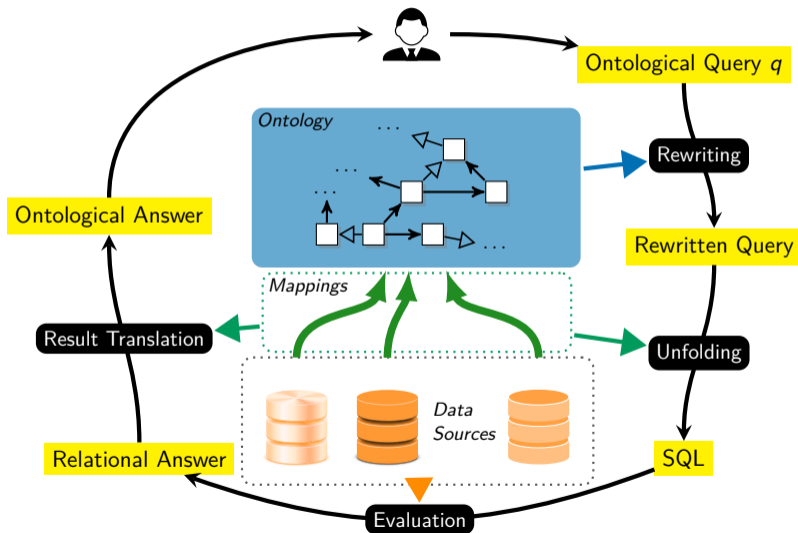
Query answering by rewriting



Query answering by rewriting



Query answering by rewriting



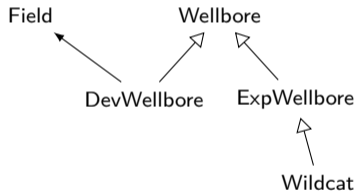
Query answering by rewriting (example)

Database:

```
wlb_dev(name, ...)  
wlb_exp(name, purpose, ...)
```

Query: *List all wellbores.*

Ontology:



Mappings:

```
DevWellbore(name)  $\mapsto$  SELECT name FROM wlb_dev  
ExpWellbore(name)  $\mapsto$  SELECT name FROM wlb_exp  
Wildcat(name)  $\mapsto$  SELECT name FROM wlb_exp  
                  WHERE purpose = 'WILDCAT'
```

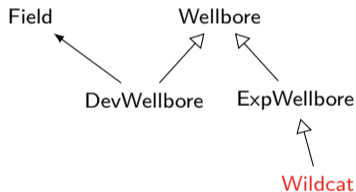
Query answering by rewriting (example)

Database:

```
wlb_dev(name, ...)  
wlb_exp(name, purpose, ...)
```

Query: *List all wellbores.*

Ontology:



Mappings:

```
DevWellbore(name)  $\mapsto$  SELECT name FROM wlb_dev  
ExpWellbore(name)  $\mapsto$  SELECT name FROM wlb_exp  
Wildcat(name)  $\mapsto$  SELECT name FROM wlb_exp  
WHERE purpose = 'WILDCAT'
```

Query answering by rewriting (example)

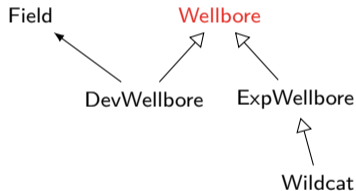
Database:

```
wlb_dev(name, ...)  
wlb_exp(name, purpose, ...)
```

Query: *List all wellbores.*

q: Wellbore(*x*)

Ontology:



Mappings:

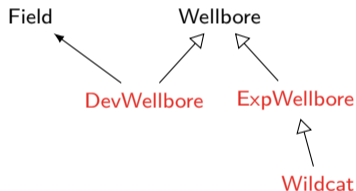
```
DevWellbore(name)  $\mapsto$  SELECT name FROM wlb_dev  
ExpWellbore(name)  $\mapsto$  SELECT name FROM wlb_exp  
Wildcat(name)  $\mapsto$  SELECT name FROM wlb_exp  
                  WHERE purpose = 'WILDCAT'
```

Query answering by rewriting (example)

Database:

```
wlb_dev(name, ...)
wlb_exp(name, purpose, ...)
```

Ontology:



Mappings:

```
DevWellbore(name)  $\mapsto$  SELECT name FROM wlb_dev
ExpWellbore(name)  $\mapsto$  SELECT name FROM wlb_exp
Wildcat(name)  $\mapsto$  SELECT name FROM wlb_exp
                     WHERE purpose = 'WILDCAT'
```

Query: *List all wellbores.*

q : Wellbore(x)

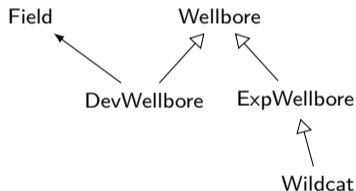
q_0 : Wellbore(x) \cup DevWellbore(x) \cup
ExpWellbore(x) \cup Wildcat(x)

Query answering by rewriting (example)

Database:

```
wlb_dev(name, ...)
wlb_exp(name, purpose, ...)
```

Ontology:



Mappings:

```
DevWellbore(name) ↦ SELECT name FROM wlb_dev
ExpWellbore(name) ↦ SELECT name FROM wlb_exp
Wildcat(name) ↦ SELECT name FROM wlb_exp
                 WHERE purpose = 'WILDCAT'
```

Query: *List all wellbores.*

q : Wellbore(x)

q_0 : Wellbore(x) \cup DevWellbore(x) \cup
ExpWellbore(x) \cup Wildcat(x)

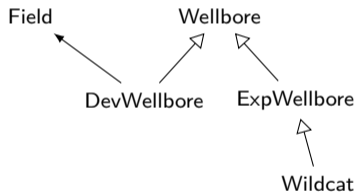
q_{SQL} : SELECT name FROM wlb_dev
UNION
SELECT name FROM wlb_exp
UNION
SELECT name FROM wlb_exp WHERE
purpose = 'WILDCAT'

Query answering by rewriting (example)

Database:

```
wlb_dev(name, ...)
wlb_exp(name, purpose, ...)
```

Ontology:



Mappings:

```
DevWellbore(name) ↦ SELECT name FROM wlb_dev
ExpWellbore(name) ↦ SELECT name FROM wlb_exp
Wildcat(name) ↦ SELECT name FROM wlb_exp
                 WHERE purpose = 'WILDCAT'
```

Query: *List all wellbores.*

q : Wellbore(x)

q_0 : Wellbore(x) \cup DevWellbore(x) \cup
ExpWellbore(x) \cup Wildcat(x)

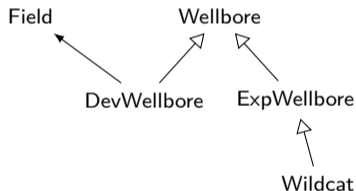
q_{SQL} : SELECT name FROM wlb_dev
UNION
SELECT name FROM wlb_exp
UNION
SELECT name FROM wlb_exp WHERE
purpose = 'WILDCAT'

Query answering by rewriting (example)

Database:

```
wlb_dev(name, ...)
wlb_exp(name, purpose, ...)
```

Ontology:



Mappings:

```
DevWellbore(name)  $\mapsto$  SELECT name FROM wlb_dev
ExpWellbore(name)  $\mapsto$  SELECT name FROM wlb_exp
Wildcat(name)  $\mapsto$  SELECT name FROM wlb_exp
                     WHERE purpose = 'WILDCAT'
```

Query: *List all wellbores.*

q : Wellbore(x)

q_0 : Wellbore(x) \cup DevWellbore(x) \cup
ExpWellbore(x) \cup Wildcat(x)

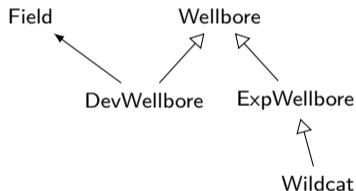
q_{SQL} : SELECT name FROM wlb_dev
UNION
SELECT name FROM wlb_exp
UNION
SELECT name FROM wlb_exp WHERE
purpose = 'WILDCAT'

Query answering by rewriting (example)

Database:

```
wlb_dev(name, ...)
wlb_exp(name, purpose, ...)
```

Ontology:



Mappings:

```
DevWellbore(name)  $\mapsto$  SELECT name FROM wlb_dev
ExpWellbore(name)  $\mapsto$  SELECT name FROM wlb_exp
Wildcat(name)  $\mapsto$  SELECT name FROM wlb_exp
                     WHERE purpose = 'WILDCAT'
```

Query: *List all wellbores.*

q : Wellbore(x)

q_0 : Wellbore(x) \cup DevWellbore(x) \cup
ExpWellbore(x) \cup Wildcat(x)

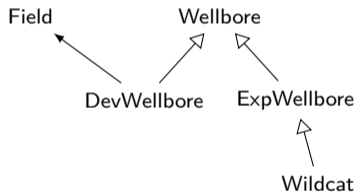
q_{SQL} : SELECT name FROM wlb_dev
UNION
SELECT name FROM wlb_exp
UNION
SELECT name FROM wlb_exp WHERE
purpose = 'WILDCAT'

Query answering by rewriting (example)

Database:

```
wlb_dev(name, ...)
wlb_exp(name, purpose, ...)
```

Ontology:



Mappings:

```
DevWellbore(name)  $\mapsto$  SELECT name FROM wlb_dev
ExpWellbore(name)  $\mapsto$  SELECT name FROM wlb_exp
Wildcat(name)  $\mapsto$  SELECT name FROM wlb_exp
                     WHERE purpose = 'WILDCAT'
```

Query: *List all wellbores.*

q : Wellbore(x)

q_0 : Wellbore(x) \cup DevWellbore(x) \cup
ExpWellbore(x) \cup Wildcat(x)

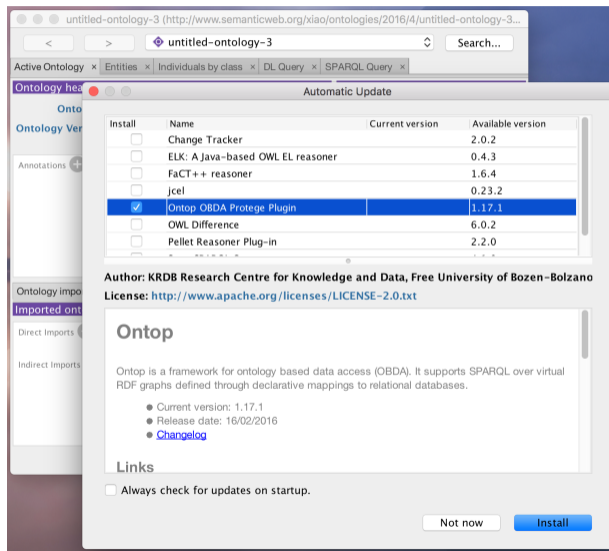
q_{SQL} : SELECT name FROM wlb_dev
UNION
SELECT name FROM wlb_exp
UNION
SELECT name FROM wlb_exp WHERE
purpose = 'WILDCAT'

q'_{SQL} : SELECT name FROM wlb_dev UNION
SELECT name FROM wlb_exp

Query answering by rewriting (tool support)

- **Ontop**: state-of-the-art OBDA system. <http://ontop.inf.unibz.it/>
- Compliant with the RDFS, OWL 2 QL, R2RML, and SPARQL standards.
- Supports all major relational DBs
 - Oracle, DB2, MS SQL Server, Postgres, MySQL, Teiid, Exareme, etc.
- **Open-source** and released under Apache 2 license
- Development of -ontop-:
 - Development started in 2009
 - -ontop- supports (essentially) all features of SPARQL 1.0 and the OWL 2 QL entailment regime of SPARQL 1.1.
 - Other features of SPARQL 1.1 (e.g., aggregates, property path queries, negation) are work in progress.

Ontop plugin available from Protégé



The screenshot shows the Protégé interface with an 'Automatic Update' dialog box open. The dialog box displays a table of available updates:

Install	Name	Current version	Available version
<input type="checkbox"/>	Change Tracker		2.0.2
<input type="checkbox"/>	ELK: A Java-based OWL EL reasoner		0.4.3
<input type="checkbox"/>	FaCT++ reasoner		1.6.4
<input type="checkbox"/>	jcel		0.23.2
<input checked="" type="checkbox"/>	Ontop OBDA Protege Plugin		1.17.1
<input type="checkbox"/>	OWL Difference		6.0.2
<input type="checkbox"/>	Pellet Reasoner Plug-in		2.2.0

Below the table, the author and license information are displayed:

Author: KRDB Research Centre for Knowledge and Data, Free University of Bozen-Bolzano
License: <http://www.apache.org/licenses/LICENSE-2.0.txt>

The dialog box also contains a description of the Ontop plugin:

Ontop

Ontop is a framework for ontology based data access (OBDA). It supports SPARQL over virtual RDF graphs defined through declarative mappings to relational databases.

- Current version: 1.17.1
- Release date: 16/02/2016
- [Changelog](#)

At the bottom, there is a checkbox for 'Always check for updates on startup.' and two buttons: 'Not now' and 'Install'.

Ontop: Mapping editor in Protégé

hospital (http://example.org/hospital) : [/Users/benjamin/bz/code/ontop-examples/swj-2015/PatientOnto.owl]

hospital (http://example.org/hospital) Search for entity

Data Properties Annotation Properties Individuals by class ontop SPARQL ontop Mappings

Active Ontology Entities Classes Object Properties

Class hierarchy: Datasource manager Mapping manager Mapping Assistant - BETA

Mapping editor:

Datasource selection
Select datasource: PatientDB

Mapping manager
Create Remove Copy Select all Select none

Patient
:db1/{patientid} a :Patient .
SELECT patientid FROM "tbl_patient"

hasName
:db1/{patientid} :hasName {name} .
SELECT patientid,name FROM "tbl_patient"

Neop
:db1/{patientid} :hasNeoplasm :db1/neoplasm/{patientid} .
SELECT patientid FROM "tbl_patient"

hasStage-IIIa
:db1/neoplasm/{patientid} :hasStage :stage-IIIa .
SELECT patientid FROM "tbl_patient" where stage=4 and type=false

Mapping count: 6 Search:

Reasoner active Show Inferences

Ontop: SPARQL query answering in Protégé

The screenshot shows the Protégé interface with the ontop query editor open. The browser address bar shows the URL: `hospital (http://example.org/hospital) : [/Users/benjamin/bz/code/ontop-examples/swj-2015/PatientOnto.owl]`. The query editor contains the following SPARQL query:

```
PREFIX : <http://example.org/hospital#>

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

Below the query editor, the execution options are set to `Execution timeout: Show: All Short IRI`. The `Attach Prefixes`, `Execute`, and `Save Changes` buttons are visible. The results table shows one result:

name
"Mary"

A hint at the bottom of the results area reads: `Hint: Try to continue scrolling down the table to retrieve more results.` The `Export to CSV...` button is also present. At the bottom of the interface, the status bar indicates `Reasoner active` and `Show Inferences` is checked.

Outline

- 1 Exposing data as RDF
- 2 Data Access in Statioil: limitations and solutions
- 3 OBDA Ingredients**
 - Overview
 - Ontology
 - Mappings
 - Query rewriting
 - Bootstrapping**
 - Visual Query Formulation
 - Optique

Bootstrapping overview

- Given a relational database (semi)automatically extracts ontological vocabulary, ontology and R2RML mappings
- Bootstrappers may also accept as input an ontology
 - R2RML mappings will link the given ontology to the database, or
 - The given ontology will be *aligned* with the bootstrapped ontology

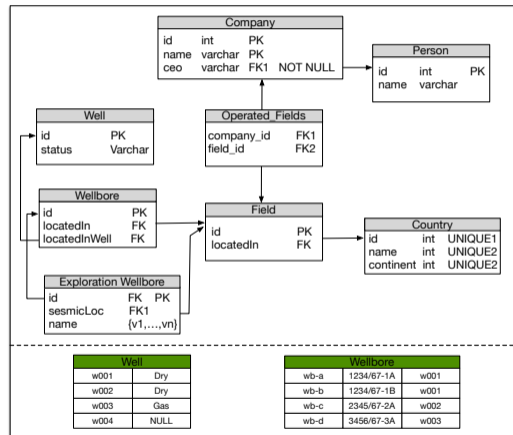
Bootstrapping overview

- Given a relational database (semi)automatically extracts ontological vocabulary, ontology and R2RML mappings
- Bootstrappers may also accept as input an ontology
 - R2RML mappings will link the given ontology to the database, or
 - The given ontology will be *aligned* with the bootstrapped ontology
- Type of mappings:
 - W3C direct mapping specification (*schema driven*)
 - To a given or bootstrapped ontology vocabulary
 - Mappings beyond direct ones (*data driven*)
 - Clusters of tuples
 - Joinable tuples

Bootstrapping: Vocabulary Generation

• W3C directives

- Tables → classes
- Foreign Keys → object properties
- Data columns → data properties
- Binary tables → fresh object properties



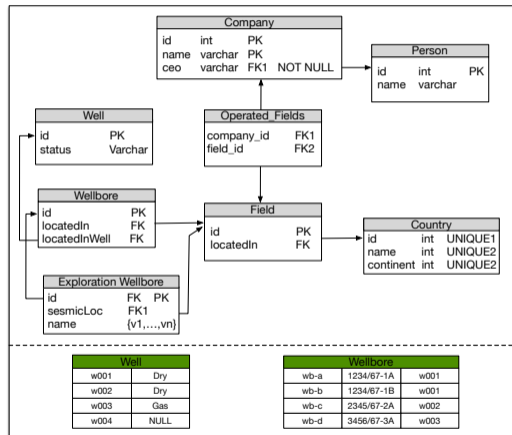
Bootstrapping: Vocabulary Generation

• W3C directives

- Tables → classes
- Foreign Keys → object properties
- Data columns → data properties
- Binary tables → fresh object properties

• Attribute naming schema:

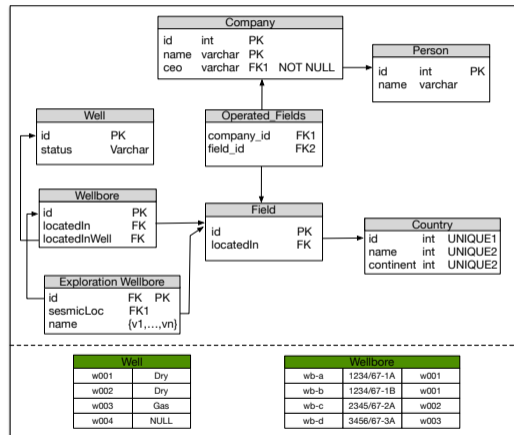
- Unique names (e.g. Person.name)
- Reusable names (e.g. name)



Bootstrapping: Axiom Generation

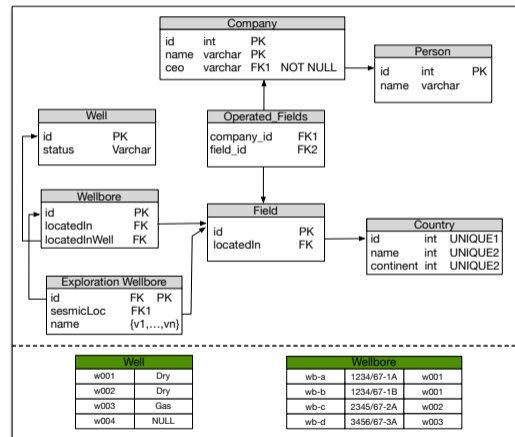
- **OWL 2 expressiveness**

- OWL 2 QL (e.g. OBDA/Optique)
- OWL 2 EL (e.g. EOLO)
- OWL 2 RL (e.g. RDFox)
- OWL 2 (e.g. PAGOdA, HermiT)



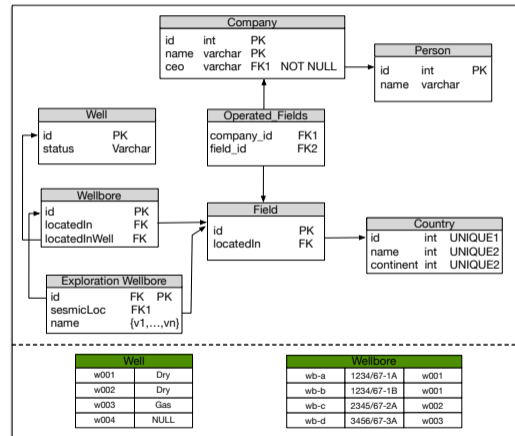
Bootstrapping: Axiom Generation

- Unique constraints
 - Person **HasKey**: id (OWL 2 RL/EL)



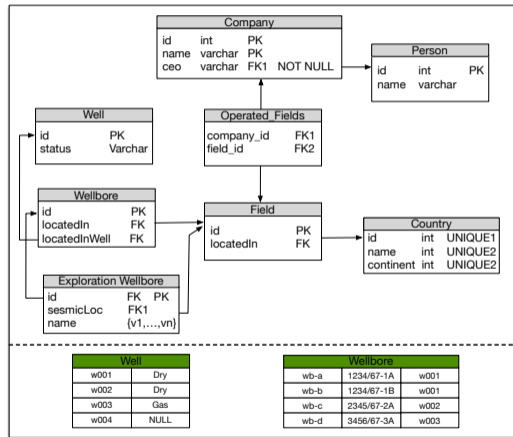
Bootstrapping: Axiom Generation

- Unique constraints
 - Person **HasKey**: id (OWL 2 RL/EL)
- Global onto. constraints
 - **Functional**: Person.name (OWL 2 RL)
 - name **Domain**: Person (all profiles)
 - Person.name **Range**: xsd:string (all profiles)



Bootstrapping: Axiom Generation

- Unique constraints
 - Person **HasKey**: id (OWL 2 RL/EL)
- Global onto. constraints
 - **Functional**: Person.name (OWL 2 RL)
 - name **Domain**: Person (all profiles)
 - Person.name **Range**: xsd:string (all profiles)
- Local onto. constraints
 - Person **subclassOf**: name **some** xsd:string (OWL 2 QL/EL)
 - Person **subclassOf**: name **only** xsd:string (OWL 2 RL)
 - Person **subclassOf**: name **exactly 1** xsd:string (OWL 2)



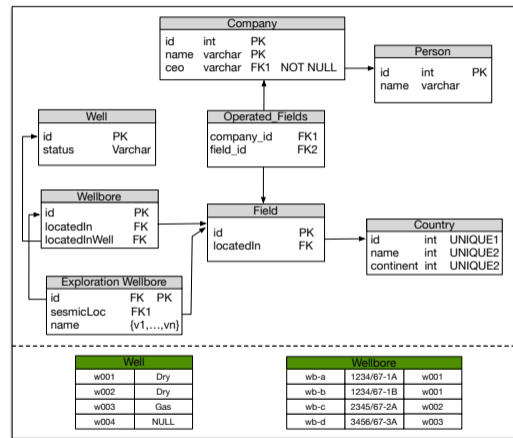
Bootstrapping: Datatypes

- Clear mapping between SQL and XML schema datatypes
- Not all XML Schema datatypes are included in OWL 2
 - *xsd:date* not in OWL 2
 - *xsd:boolean* and *xsd:double* not in OWL 2 QL/EL
- Value spaces of primitive datatypes are disjoint (e.g. *xsd:double* and *xsd:decimal*)

Bootstrapping: Datatypes

- Clear mapping between SQL and XML schema datatypes
- Not all XML Schema datatypes are included in OWL 2
 - *xsd:date* not in OWL 2
 - *xsd:boolean* and *xsd:double* not in OWL 2 QL/EL
- Value spaces of primitive datatypes are disjoint (e.g. *xsd:double* and *xsd:decimal*)
- ✗ Problems when materializing the data or using the ontology for virtual access to data
- ✓ Solution: *rdfs:Literal*

Bootstrapping: Taxonomy Generation



Bootstrapping: Taxonomy Generation

- **Data driven**

- Clusters of tuples
- Joinable tuples
- e.g. Well_Dry **SubclassOf:** Well (w001, w002)
- e.g. Well_with_Wellbore **SubclassOf:** Well (w001, w002, w003)

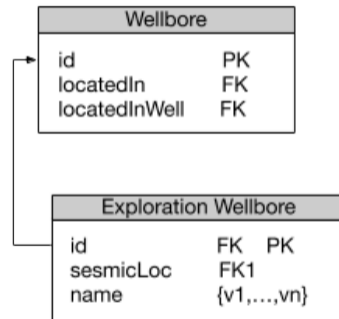
Well	
w001	Dry
w002	Dry
w003	Gas
w004	NULL

Wellbore		
wb-a	1234/67-1A	w001
wb-b	1234/67-1B	w001
wb-c	2345/67-2A	w002
wb-d	3456/67-3A	w003

Bootstrapping: Taxonomy Generation

- **Schema driven**

- A single-column Foreign Key and Primary key
- e.g. Exploration_Wellbore **SubclassOf:** Wellbore



Tool support and Lessons learnt

- **BootOX**: <http://www.cs.ox.ac.uk/isg/tools/BootOX/>
- **Feedback from use cases and evaluation...**
 - ✓ Good as a first approximation of the ontology and mappings
 - ✓ Competitive results in (academic) benchmarks
 - ✗ For the largest Statoil datasources, the solution is far from perfect
 - ✗ Ontology close to the original database
 - ✗ Large amount of ontology entities and R2RML mappings
 - ✓ Implementation of an incremental/interactive bootstrapping

Tool support and Lessons learnt

- **BootOX**: <http://www.cs.ox.ac.uk/isg/tools/BootOX/>
- **Feedback from use cases and evaluation...**
 - ✓ Good as a first approximation of the ontology and mappings
 - ✓ Competitive results in (academic) benchmarks
 - ✗ For the largest Statoil datasources, the solution is far from perfect
 - ✗ Ontology close to the original database
 - ✗ Large amount of ontology entities and R2RML mappings
 - ✓ Implementation of an incremental/interactive bootstrapping
- **Future work...**
 - New ways to exploit the data and schema (e.g. views)

Outline

- 1 Exposing data as RDF
- 2 Data Access in Statioil: limitations and solutions
- 3 OBDA Ingredients**
 - Overview
 - Ontology
 - Mappings
 - Query rewriting
 - Bootstrapping
 - Visual Query Formulation**
 - Optique

Visual query formulation (OptiqueVQS)

The screenshot displays the OptiqueVQS interface for visual query formulation. The main workspace shows a query graph with three nodes:

- W3**: An "Untitled query" node with the text "Please provide a description here...".
- kernel**: A "Wellbore name(o)" node.
- inverted**: A node with the relationship "(inv) hasWe...".
- pivot**: An orange node with the relationship "Well dateSyncNPD(o) wellType(c)".
- relation**: A node with the relationship "drillingOpe...".

Below the workspace is a toolbar with the following actions: Delete Node, Undo, Redo, New Query, Save Query, Stored Queries, Q-Config, SPARQL Query, and Run Query.

The bottom section is divided into two panels:

- Left Panel (W1)**: A "Well" property browser showing a list of properties: "ProductionLicenceAreaPerBlock (inv) country", "Company wellOperatedBy", "License wellLicense", and "WellboreStratigraphicCoreSet". The "Company" property is expanded, showing "range" and "relation" options.
- Right Panel (W2)**: An "output" configuration panel for the "dateSyncNPD" property. It shows two input fields with the format "mm/dd/yyyy, --:-- --". Below this, the "wellType" property is selected with a "constraint" button, and the value "DEVELOPMENT" is shown.

Visual query formulation (OptiqueVQS)

The screenshot displays the OptiqueVQS interface for the 'Well' ontology. The top section is a SPARQL query editor with the following query:

```
PREFIX ns3: <http://www.optique-project.eu/well-ontology/>

SELECT ?c1 ?a3 ?c2 ?a1 ?c3 ?a4 WHERE {
  ?c1 ns1:type ns2:Wellbore.
  ?c2 ns1:type ns3:Well.
  ?c3 ns1:type ns2:Company.
  ?c1 ^ns3:hasWellbore ?c2.
  ?c1 ns2:drillingOperatorCompany ?c3.
  ?c1 ns2:name ?a3.
  ?c2 ns2:dateSyncNPD ?a1.
  ?c2 ns2:wellType ?a2.
  ?c3 ns2:name ?a4.
  FILTER(regex(?a2, "DEVELOPMENT", "i")).
```

Below the editor is a toolbar with buttons: Delete Node, Undo, Redo, New Query, Save Query, Stored Queries, Q-Config, SPARQL Query, and Run Query.

The bottom section is a visual query builder with two panes. The left pane, titled 'Well', shows a list of classes with icons and right-pointing arrows:

- ProductionLicenceAreaPerBlock** (inv) country
- Company** wellOperatedBy
- License** wellLicense

The right pane, also titled 'Well', features a search bar and a list of properties with input fields:

- dateSyncNPD**: mm/dd/yyyy, --:-- --
- wellType**: mm/dd/yyyy, --:-- --
- wellType**: DEVELOPMENT

Visual query formulation (OptiqueVQS)

The image displays the OptiqueVQS interface, which is used for visual query formulation. The main window is titled "Field" and contains a "Map" view. The map shows a geographical area, likely Denmark, with a black pin icon labeled "WS" indicating a well location. The map includes a compass, a zoom slider, and a scale bar. The selected location is identified as "VALHALL" with coordinates 13.24524, 56.21923. The interface also features a search bar, a "Delete Node" button, an "Undo" button, and a "SPARQL Query" input field. A "map activation" button is visible in the bottom right corner. The query graph on the right shows a "Field" node connected to a "Well" node, which is further connected to a "Dimension" node and a "Place" node.

Field

Map

Well (inv) inField

Dimension hasDimension

Place locatedIn

Selected: VALHALL

Send selection

SPARQL Query Run Query

map activation

Visual query formulation (OptiqueVQS)

An example query

```
PREFIX ns1: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ns2: <http://www.siemens.com/demo#>
PREFIX ns3: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?c1 ?a1 WHERE {
  ?c1 ns1:type ns2:Sensor.
  ?c1 ns3:label ?a1.
}
```

query management

Total (2) Saved (2) Draft (0)

Filter...

- An example query

A query with aggregation

Load Delete

+ Another query

Example Results

Full result set example results Columns to display...

Sensor_c1	label_a1	
Go to resource	Exit Temperature	Sum
Go to resource	Vane Feedback	Avg
Go to resource	Exit Temperature	Max
Go to resource	Blow Off Valve Position	Min
Go to resource	Temperature Fire	Count
Go to resource	?	S.Up
Go to resource	Exit Temperature	S.Down

aggregate functions

W4

Outline

- 1 Exposing data as RDF
- 2 Data Access in Statioil: limitations and solutions
- 3 OBDA Ingredients**
 - Overview
 - Ontology
 - Mappings
 - Query rewriting
 - Bootstrapping
 - Visual Query Formulation
 - Optique**

Optique infrastructure

- Optique project:
 - <http://optique-project.eu/>
- OptiqueVQS can be (partially) tested online
 - <http://sws.ifi.uio.no/project/optique-vqs/>
- What is next?
 - *SIRIUS*: <http://sirius-labs.no/>

Questions?

Ernesto Jiménez-Ruiz (ernestoj@ifi.uio.no)
Office hours: from 9:00 to 16:00 at OJD 8165

Acknowledgements

- -ontop- team (Diego Calvanese, Benjamin Cogrel, Guohui Xiao, etc.) and Dag Hovland for sharing their wonderful slides
 - <http://ontop.inf.unibz.it/ekaw-2016-tutorial/>
- Ahmet Soylu: main person behind OptiqueVQS
- Optique and SIRIUS partners