COMP718: Ontologies and Knowledge Bases
Lecture 9: Ontology/Conceptual Model based Data Access

Maria Keet
email: keet@ukzn.ac.za
home: http://www.meteck.org

School of Mathematics, Statistics, and Computer Science
University of KwaZulu-Natal, South Africa

10 April 2012
Outline

1. OBDA Options

2. Some technical details
   - Introduction
   - The ontology language
   - The mapping layer
     - ‘Impedance’ mismatch
     - Mapping assertions
   - Query answering
An ontology with a very large ABox (intro last week)

- Scaling up to realistic size knowledge base handling large amounts of data
- To realise this, we need
  - A language of relatively low computational complexity
  - A way to store large amounts of data
  - Some mechanism to link up the previous two ingredients
  - Query (and reason over) the combination of the previous three
- Use the “Ontology-Based Data Access” (OBDA) approach
  - with the “ontology” in OBDA just a DL knowledge base
  - Most examples and use cases: the ‘ontology’ is a DL-formalised conceptual data model
- Example application with the WONDER system
An ontology with a very large ABox (this week)

⇒ What are the options to link an ontology to large amounts of data?
  - Two principal options (in KR view): query rewriting and data completion
  - Several implementation infrastructures; ‘external ABox’ most popular (realised with RDBMS or RDF Triple store)

⇒ What is there behind the scenes for the non-graphical OBDA-part in WONDER and the OBDA systems you set up in the lab?
Outline

1. OBDA Options

2. Some technical details
   - Introduction
   - The ontology language
   - The mapping layer
     - ‘Impedance’ mismatch
     - Mapping assertions
   - Query answering
OBDA options

- KR perspective (with OWA): query rewriting vs data completion
- DB perspective (with CWA): we probably won’t cover this in the lecture
- See slides obda-slides2012TomanCOMP718ukzn.pdf
Outline

1 OBDA Options

2 Some technical details
   - Introduction
   - The ontology language
   - The mapping layer
     - ‘Impedance’ mismatch
     - Mapping assertions
   - Query answering
Introduction

Linking ontologies to relational data

- Ontology-Based Data Access systems (static components)
  - An ontology language
  - A mapping language
  - The data

- Query answering in Ontology-Based Data Access systems
  - Reasoning over the TBox
  - Query rewriting
  - Query unfolding
  - Relational database technology

*These slides are based on Calvanese’s MOSS’09 slides, which also will be made available*

---

1 More precisely: “Option I, v1.0” mentioned in David Toman’s slides.
Definition (Ontology-Based Data Access system)

An OBDA system is a triple $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$, where

- $\mathcal{T}$ is a TBox
- $\mathcal{D}$ is a relational database
- $\mathcal{M}$ is a set of mapping assertions between $\mathcal{T}$ and $\mathcal{D}$

**Note:** this is for the current system, but one could conceive of a system that has an RDF triple store as $\mathcal{D}$
In the traditional DL setting, it is assumed that the data is maintained in the ABox of the ontology, meaning:

- The ABox is perfectly compatible with the TBox:
  - The vocabulary of concepts, roles, and attributes is the one used in the TBox
  - The ABox stores abstract objects, and these objects and their properties are those returned by queries over the ontology

- Other ways to manage the ABox from an implementation point of view:
  - Description Logics reasoners maintain the ABox is main-memory data structures (recollect the 4 GB HGT-DB)
  - Hence, when an ABox becomes large, managing it in secondary storage may be required, but this is again handled directly by the reasoner
In the traditional DL setting, it is assumed that the data is maintained in the ABox of the ontology, meaning:

- **The ABox is perfectly compatible with the TBox:**
  - The vocabulary of concepts, roles, and attributes is the one used in the TBox
  - The ABox stores abstract objects, and these objects and their properties are those returned by queries over the ontology

- **Other ways to manage the ABox from an implementation point of view:**
  - Description Logics reasoners maintain the ABox is main-memory data structures (recollect the 4 GB HGT-DB)
  - Hence, when an ABox becomes large, managing it in secondary storage may be required, but this is again handled directly by the reasoner
**Introduction**

\[ \mathcal{D} = \text{Relational database as ABox} \]

- In addition to ABox scalability, there are other reasons to realise the ABox with \( \mathcal{D} \):
  - When we have no direct control over the data since it belongs to some external organization, which controls the access to it
  - When multiple data sources need to be accessed, such as in Information Integration

- Deal with such a situation by keeping the data in the external (relational) storage, and performing query answering by leveraging the capabilities of the relational engine

- New problems:
  - The so-called impedance mismatch between values in the relational database and the objects that the ABox expects
  - How to link the TBox to the “ABox” that is realised as a \( \mathcal{D} \)?
\( D = \) Relational database as ABox

- In addition to ABox scalability, there are other reasons to realise the ABox with \( D \):
  - When we have no direct control over the data since it belongs to some external organization, which controls the access to it
  - When multiple data sources need to be accessed, such as in Information Integration

- Deal with such a situation by keeping the data in the external (relational) storage, and performing query answering by leveraging the capabilities of the relational engine

- New problems:
  - The so-called impedance mismatch between values in the relational database and the objects that the ABox expects
  - How to link the TBox to the “ABox” that is realised as a \( D \)?
In addition to ABox scalability, there are other reasons to realise the ABox with $\mathcal{D}$:

- When we have no direct control over the data since it belongs to some external organization, which controls the access to it
- When multiple data sources need to be accessed, such as in Information Integration

Deal with such a situation by keeping the data in the external (relational) storage, and performing query answering by leveraging the capabilities of the relational engine

New problems:

- The so-called impedance mismatch between values in the relational database and the objects that the ABox expects
- How to link the TBox to the “ABox” that is realised as a $\mathcal{D}$?
The DL-Lite family

- A family of DLs optimized according to the tradeoff between expressive power and complexity of query answering, with emphasis on data
- Carefully designed to have nice computational properties for answering UCQs (i.e., computing certain answers):
  - The same complexity as relational databases
  - Query answering can be delegated to a relational DB engine
  - The DLs of the DL-Lite family are essentially the maximally expressive ontology languages enjoying these nice computational properties
- Introduction of DL-Lite₆, a member of the DL-Lite family, essentially corresponds to OWL2 QL\(^2\)

\(^2\)Actually, the current OBDA implementation can handle DL-Lite₆, and all DL-Lite languages adhere to the
The ontology language

**DL-Lite** \( \mathcal{R} \) (compact DL notation of OWL 2 QL)

TBox assertions:
- Concept inclusion assertions: \( Cl \sqsubseteq Cr \), with:
  \[
  Cl \rightarrow A \mid \exists Q \\
  Cr \rightarrow A \mid \exists Q \mid \neg A \mid \neg \exists Q \\
  Q \rightarrow P \mid P^- 
  \]
- Property inclusion assertions: \( Q \sqsubseteq R \), with:
  \[
  R \rightarrow Q \mid \neg Q 
  \]

ABox assertions: \( A(c), \ P(c_1, c_2) \), with \( c_1, c_2 \) constants

**Note:** DL-Lite\( \mathcal{R} \) can be straightforwardly adapted to distinguish also between object and data properties (attributes).
### DL-Lite\(\mathcal{R}\) (compacter DL notation of OWL 2 QL)

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntax</th>
<th>Example</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic conc.</td>
<td>(A)</td>
<td>Doctor</td>
<td>(A^I \subseteq \Delta^I)</td>
</tr>
<tr>
<td>exist. restr.</td>
<td>(\exists Q)</td>
<td>(\exists \text{child}^\neg)</td>
<td>({d \mid \exists e. (d, e) \in Q^I})</td>
</tr>
<tr>
<td>at. conc. neg.</td>
<td>(\neg A)</td>
<td>(\neg \text{Doctor})</td>
<td>(\Delta^I \setminus A^I)</td>
</tr>
<tr>
<td>conc. neg.</td>
<td>(\neg \exists Q)</td>
<td>(\neg \exists \text{child})</td>
<td>(\Delta^I \setminus (\exists Q)^I)</td>
</tr>
<tr>
<td>atomic role</td>
<td>(P)</td>
<td>child</td>
<td>(P^I \subseteq \Delta^I \times \Delta^I)</td>
</tr>
<tr>
<td>inverse role</td>
<td>(P^\neg)</td>
<td>(\text{child}^\neg)</td>
<td>({(o, o') \mid (o', o) \in P^I})</td>
</tr>
<tr>
<td>role negation</td>
<td>(\neg Q)</td>
<td>(\neg \text{manages})</td>
<td>((\Delta^I \times \Delta^I) \setminus Q^I)</td>
</tr>
<tr>
<td>conc. incl.</td>
<td>(Cl \subseteq Cr)</td>
<td>Father (\subseteq \exists \text{child})</td>
<td>(Cl^I \subseteq Cr^I)</td>
</tr>
<tr>
<td>role incl.</td>
<td>(Q \subseteq R)</td>
<td>hasFather (\subseteq \text{child}^\neg)</td>
<td>(Q^I \subseteq R^I)</td>
</tr>
<tr>
<td>mem. asser.</td>
<td>(A(c))</td>
<td>Father(bob)</td>
<td>(c^I \in A^I)</td>
</tr>
<tr>
<td>mem. asser.</td>
<td>(P(c_1, c_2))</td>
<td>child(bob, ann)</td>
<td>((c_1^I, c_2^I) \in P^I)</td>
</tr>
</tbody>
</table>
The ontology language

**DL-Lite** (compacter DL notation of OWL 2 QL)

<table>
<thead>
<tr>
<th>ISA between classes</th>
<th>$A_1 \subseteq A_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disjointness between classes</td>
<td>$A_1 \nsubseteq \neg A_2$</td>
</tr>
<tr>
<td>Domain and range of properties</td>
<td>$\exists P \subseteq A_1$ $\exists P^- \subseteq A_2$</td>
</tr>
<tr>
<td>Mandatory participation ($\text{min card} = 1$)</td>
<td>$A_1 \subseteq \exists P$ $A_2 \subseteq \exists P^-$</td>
</tr>
<tr>
<td>ISA between properties</td>
<td>$Q_1 \subseteq Q_2$</td>
</tr>
<tr>
<td>Disjointness between properties</td>
<td>$Q_1 \subseteq \neg Q_2$</td>
</tr>
</tbody>
</table>

**Note:** DL-Lite cannot capture completeness of a hierarchy. This would require **disjunction** (i.e., OR).

**Note 2:** DL-Lite cannot capture **functionality** on roles ($\text{max card} = 1$)
Relational database as ABox

- Sources store data, which is constituted by values taken from concrete domains, such as strings, integers, codes, ...
- Instances of concepts and relations in an ontology are (abstract) objects
- Solution:
  - Specify how to construct from the data values in the relational sources the (abstract) objects that populate the ABox of the ontology
  - Embed this specification in the mappings between the data sources and the ontology
- Use a virtual ABox, where the objects are not materialized
Sources store data, which is constituted by values taken from concrete domains, such as strings, integers, codes, ...

Instances of concepts and relations in an ontology are (abstract) objects

Solution:

- Specify how to construct from the data values in the relational sources the (abstract) objects that populate the ABox of the ontology
- Embed this specification in the mappings between the data sources and the ontology

Use a virtual ABox, where the objects are not materialized
Sources store data, which is constituted by values taken from concrete domains, such as strings, integers, codes, ...  

Instances of concepts and relations in an ontology are (abstract) objects  

Solution:  
- Specify how to construct from the data values in the relational sources the (abstract) objects that populate the ABox of the ontology  
- Embed this specification in the mappings between the data sources and the ontology  

Use a virtual ABox, where the objects are not materialized
Solution to the impedance mismatch

- Define a mapping language that allows for specifying how to transform data into abstract objects, where
  - Each mapping assertion maps a query that retrieves values from a data source to a set of atoms specified over the ontology
- Basic idea: use Skolem functions in the atoms over the ontology to “generate” the objects from the data values
- Semantics of mappings:
  - Objects are denoted by terms (of exactly one level of nesting)
  - Different terms denote different objects (i.e., we make the unique name assumption on terms)
Actual data is stored in a DB:
- An employee is identified by her SSN.
- A project is identified by its name.

\[D_1[SSN: \text{String}, PrName: \text{String}]\]
Employees and projects they work for

\[D_2[Code: \text{String}, Salary: \text{Int}]\]
Employee’s code with salary

\[D_3[Code: \text{String}, SSN: \text{String}]\]
Employee’s Code with SSN

Intuitively:
- An employee should be created from her SSN: \text{pers}(SSN)
- A project should be created from its name: \text{proj}(PrName)
The mapping layer

Associate objects in the ontology to data in the tables

- Introduce an alphabet $\Lambda$ of function symbols, each with an associated arity
- Use value constants from an alphabet $\Gamma_V$ to denote values
- Use object terms instead of object constants to denote objects: and object term has the form $f(d_1, \ldots, d_n)$ with $f \in \Lambda$, and each $d_i$ is a value constant in $\Gamma_V$

**Example**

- If a person is identified by her SSN, we can introduce a function symbol $\text{pers}/1$. If NRM18JUL18 is a SSN, then $\text{pers}(\text{NRM18JUL18})$ denotes a person.
- If a person is identified by her name and dateOfBirth, we can introduce a function symbol $\text{pers}/2$. Then $\text{pers}(\text{Mandela, 18/07/18})$ denotes a person.
The mapping layer

Associate objects in the ontology to data in the tables

- Introduce an alphabet \( \Lambda \) of function symbols, each with an associated arity
- Use value constants from an alphabet \( \Gamma_V \) to denote values
- Use object terms instead of object constants to denote objects: and object term has the form \( f(d_1, \ldots, d_n) \) with \( f \in \Lambda \), and each \( d_i \) is a value constant in \( \Gamma_V \)

Example

- If a person is identified by her SSN, we can introduce a function symbol \( \text{pers}/1 \). If \( \text{NRM18JUL18} \) is a SSN, then \( \text{pers}(\text{NRM18JUL18}) \) denotes a person.
- If a person is identified by her name and dateOfBirth, we can introduce a function symbol \( \text{pers}/2 \). Then \( \text{pers}(\text{Mandela, 18/07/18}) \) denotes a person.
The mapping layer

Mapping assertions, formally

- Mapping assertions are used to extract the data from the DB to populate the ontology
- Use of **variable terms**, which are like object terms, but with variables instead of values as arguments of the functions

**Definition (Mapping assertion between a database and a TBox)**

A mapping assertion between a database $\mathcal{D}$ and a TBox $\mathcal{T}$ has the form

$$\Phi \leadsto \Psi$$

where

- $\Phi$ is an arbitrary SQL query of arity $n > 0$ over $\mathcal{D}$;
- $\Psi$ is a conjunctive query over $\mathcal{T}$ of arity $n' > 0$ without non-distinguished variables, possibly involving variable terms.
Mapping assertions, formally

- Mapping assertions are used to extract the data from the DB to populate the ontology.
- Use of **variable terms**, which are like object terms, but with variables instead of values as arguments of the functions.

**Definition (Mapping assertion between a database and a TBox)**

A mapping assertion between a database $\mathcal{D}$ and a TBox $\mathcal{T}$ has the form

$$\Phi \rightsquigarrow \Psi$$

where

- $\Phi$ is an arbitrary SQL query of arity $n > 0$ over $\mathcal{D}$;
- $\Psi$ is a conjunctive query over $\mathcal{T}$ of arity $n' > 0$ without non-distinguished variables, possibly involving variable terms.
The mapping layer

Example

**D₁** [**SSN**: String, **PrName**: String]
Employees and Projects they work for

**D₂** [**Code**: String, **Salary**: Int]
Employee’s code with salary

**D₃** [**Code**: String, **SSN**: String]
Employee’s code with SSN

\[...

\]

\[m₁: \text{SELECT } \text{SSN, PrName} \quad \leadsto \quad \text{Employee(pers(SSN))}, \\
\text{Project(proj(PrName))}, \\
\text{proj(PrName, PrName)}, \\
\text{worksFor(pers(SSN, proj(PrName))}) \]

\[m₂: \text{SELECT } \text{SSN, Salary} \quad \leadsto \quad \text{Employee(pers(SSN))}, \\
\text{salary(pers(SSN, Salary))} \]

FROM D₁
FROM D₂, D₃
WHERE D₂.Code = D₃.Code
Definition (Mapping assertion in \( M \) in an OBDA system)

A mapping assertion between a database \( D \) and a TBox \( T \) in \( M \) has the form

\[
\Phi(\vec{x}) \leadsto \Psi(\vec{t}, \vec{y})
\]

where

- \( \Phi \) is an arbitrary SQL query of arity \( n > 0 \) over \( D \);
- \( \Psi \) is a conjunctive query over \( T \) of arity \( n' > 0 \) without non-distinguished variables;
- \( \vec{x}, \vec{y} \) are variables with \( \vec{y} \subseteq \vec{x} \);
- \( \vec{t} \) are variable terms of the form \( f(\vec{z}) \), with \( f \in \Lambda \) and \( \vec{z} \subseteq \vec{x} \).
The mapping layer

Semantics of mappings

Intuitively: $\mathcal{I}$ satisfies $\Phi \leadsto \Psi$ with respect to $\mathcal{D}$ if all facts obtained by evaluating $\Phi$ over $\mathcal{D}$ and then propagating answers to $\Psi$, hold in $\mathcal{I}$.

Definition (Satisfaction of a mapping assertion with respect to a database)

An interpretation $\mathcal{I}$ satisfies a mapping assertion $\Phi(\bar{x}) \leadsto \Psi(\bar{t}, \bar{y})$ in $\mathcal{M}$ with respect to a database $\mathcal{D}$, if for each tuple of values $\bar{v} \in \text{Eval}(\Phi, \mathcal{D})$, and for each ground atom in $\Psi[\bar{x}/\bar{v}]$, we have that:

- If the ground atom is $A(s)$, then $s^\mathcal{I} \in A^\mathcal{I}$;
- If the ground atom is $P(s_1, s_2)$, then $(s_1^\mathcal{I}, s_2^\mathcal{I}) \in P^\mathcal{I}$.

$\text{Eval}(\Phi, \mathcal{D})$ denotes the result of evaluating $\Phi$ over $\mathcal{D}$, $\Psi[\bar{x}/\bar{v}]$ denotes $\Psi$ where each $x_i$ is substituted with $v_i$. 
Semantics of an OBDA system

Definition (Model of an OBDA system)

An interpretation $\mathcal{I}$ is a model of $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ if:

- $\mathcal{I}$ is a model of $\mathcal{T}$;
- $\mathcal{I}$ satisfies $\mathcal{M}$ with respect to $\mathcal{D}$, i.e., every assertion in $\mathcal{M}$ w.r.t. $\mathcal{D}$.

An OBDA system $\mathcal{O}$ is satisfiable if it admits at least one model
The mapping layer

Semantics of an OBDA system

Definition (Model of an OBDA system)

An interpretation $\mathcal{I}$ is a model of $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ if:

- $\mathcal{I}$ is a model of $\mathcal{T}$;
- $\mathcal{I}$ satisfies $\mathcal{M}$ with respect to $\mathcal{D}$, i.e., every assertion in $\mathcal{M}$ w.r.t. $\mathcal{D}$.

An OBDA system $\mathcal{O}$ is satisfiable if it admits at least one model
Two approaches for query answering over $O$

- **Bottom-up approach:**
  - Explicitly construct an ABox $A_{M,D}$ using $D$ and $M$, and compute the certain answers over $\langle T, A_{M,D} \rangle$
  - Conceptually simpler, but less efficient (PTime in the data).

- **Top-down approach**
  - Unfold the query w.r.t. $M$ and generate a query over $D$.
  - Is more sophisticated, but also more efficient

- OBDA with QUONTO/Quest uses the top-down approach
Top-down approach to query answering, intuition

To be able to deal with data efficiently, we need to separate the contribution of $A$ from the contribution of $q$ and $T$. 
Top-down approach to query answering, intuition

Diagram:
- $q$ → Perfect reformulation (under OWA) → $r_{q,T}$ → Query evaluation (under CWA) → $\text{cert}(q, (T, A))$
Top-down approach to query answering

- **Reformulation**: compute the perfect reformulation (rewriting), $q_{pr} = \text{PerfectRef}(q, \mathcal{T}_P)$, of the original query $q$ using the inclusion assertions of the TBox $\mathcal{T}$ so that we have a UCQ.

- **Unfolding**: compute a new query $q_{unf}$ from $q_{pr}$ by using the (split version of) the mappings in $\mathcal{M}$
  - Each atom in $q_{pr}$ that unifies with an atom in $\Psi$ is substituted with the corresponding query $\Phi$ over the database
  - The unfolded query is such that $\text{Eval}(q_{unf}, \mathcal{D}) = \text{Eval}(q_{pr}, \mathcal{A}_\mathcal{M}, \mathcal{D})$

- **Evaluation**: delegate the evaluation of $q_{unf}$ to the relational DBMS managing $\mathcal{D}$

More examples, rewriting rules and algorithm are described on pp290-297 of the MOSS’09 slides, and more details on unfolding are on pp248-251 of the MOSS’09 slides.
Consider the query  

\[ q(x) \leftarrow \text{worksFor}(x, y) \]

the perfect rewriting is

\[ r_{q,T} = q(x) \leftarrow \text{worksFor}(x, y) \]

\[ q(x) \leftarrow \text{Employee}(x) \]
Example

\[ D_1[SSN: \text{String}, \ PrName: \text{String}] \]
Employees and Projects they work for

\[ D_2[Code: \text{String}, \ Salary: \text{Int}] \]
Employee’s code with salary

\[ D_3[Code: \text{String}, \ SSN: \text{String}] \]
Employee’s code with SSN

\[ m_1: \text{SELECT SSN, PrName FROM } D_1 \]
\[ \leadsto \text{Employee(pers(SSN)), Project(proj(PrName)), projectName(proj(PrName), PrName), worksFor(pers(SSN), proj(PrName))} \]

\[ m_2: \text{SELECT SSN, Salary FROM } D_2, D_3 \]
WHERE \text{D}_2.\text{Code} = \text{D}_3.\text{Code}
\[ \leadsto \text{Employee(pers(SSN)), salary(pers(SSN), Salary)} \]
To compute $\text{unfold}(r_q,T)$, we first split $\mathcal{M}$ as follows (always possible, since queries in the right-hand side of assertions in $\mathcal{M}$ are without non-distinguished variables):

$M_{1,1}$: \begin{align*}
\text{SELECT} & \text{ SSN, PrName} \\
\text{FROM} & \text{ D}_1
\end{align*} \quad \leadsto \text{ Employee(pers(SSN))}

$M_{1,2}$: \begin{align*}
\text{SELECT} & \text{ SSN, PrName} \\
\text{FROM} & \text{ D}_1
\end{align*} \quad \leadsto \text{ Project(proj(PrName))}

$M_{1,3}$: \begin{align*}
\text{SELECT} & \text{ SSN, PrName} \\
\text{FROM} & \text{ D}_1
\end{align*} \quad \leadsto \text{ projectName(proj(PrName), PrName)}

$M_{1,4}$: \begin{align*}
\text{SELECT} & \text{ SSN, PrName} \\
\text{FROM} & \text{ D}_1
\end{align*} \quad \leadsto \text{ workFor(pers(SSN), proj(PrName))}

$M_{2,1}$: \begin{align*}
\text{SELECT} & \text{ SSN, Salary} \\
\text{FROM} & \text{ D}_2, D_3 \\
\text{WHERE} & \text{ D}_2.\text{Code} = D_3.\text{Code}
\end{align*} \quad \leadsto \text{ Employee(pers(SSN))}

$M_{2,2}$: \begin{align*}
\text{SELECT} & \text{ SSN, Salary} \\
\text{FROM} & \text{ D}_2, D_3 \\
\text{WHERE} & \text{ D}_2.\text{Code} = D_3.\text{Code}
\end{align*} \quad \leadsto \text{ salary(pers(SSN), Salary)}
Then, we unify each atom of the query

\[ r_{q,T} = q(x) \leftarrow \text{worksFor}(x,y) \]
\[ q(x) \leftarrow \text{Employee}(x) \]

with the right-hand side of the assertion in the split mapping, and substitute such atom with the left-hand side of the mapping

\[ q(\text{pers(SSN)}) \leftarrow \text{SELECT SSN, PrName} \]
\[ \quad \text{FROM } D_1 \]
\[ q(\text{pers(SSN)}) \leftarrow \text{SELECT SSN, Salary} \]
\[ \quad \text{FROM } D_2, D_3 \]
\[ \quad \text{WHERE } D_2.\text{CODE} = D_3.\text{CODE} \]

The construction of object terms can be pushed into the SQL query, by resorting to SQL functions to manipulate strings (e.g., string concat).
SELECT concat(concat('pers (',SSN),'))
FROM D_1
UNION
SELECT concat(concat('pers (',SSN),'))
FROM D_2, D_3
Implementation of top-down approach to query answering

To generate an SQL query, one can follow different strategies:

- Substitute each view predicate in the unfolded queries with the corresponding SQL query over the source:
  - joins are performed on the DB attributes
  - does not generate doubly nested queries
  - the number of unfolded queries may be exponential

- Construct for each atom in the original query a new view. This view takes the union of all SQL queries corresponding to the view predicates, and constructs also the Skolem terms
  - avoids exponential blow-up of the resulting query, since the union (of the queries coming from multiple mappings) is done before the joins
  - joins are performed on Skolem terms
  - generates doubly nested queries

Which method is better, depends on various parameters
Summary

1 OBDA Options

2 Some technical details
   - Introduction
   - The ontology language
   - The mapping layer
     - ‘Impedance’ mismatch
     - Mapping assertions
   - Query answering