Ontology and Database Systems: Knowledge Representation and Ontologies

Part 4: Ontology Based Data Access

Diego Calvanese

Faculty of Computer Science
European Master in Computational Logic

A.Y. 2014/2015

unibz
Fakultät für Informatik
Facoltà di Scienze e Tecnologie informatiche
Faculty of Computer Science

Part 4

Ontology-based data access

Outline of Part 4

unibz

Outline of Part 4

Outline of Part 4

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 data complexity as relational databases.
  - In fact, 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.
- Captures conceptual modeling formalism.

The *DL-Lite* family provides new foundations for Ontology-Based Data Access.

Basic features of *DL-Lite*$_{\mathcal{A}}$

DL-Lite$_{\mathcal{A}}$ is an expressive member of the *DL-Lite* family.

- Takes into account the distinction between **objects** and **values**:
  - Objects are elements of an abstract interpretation domain.
  - Values are elements of concrete data types, such as integers, strings, ecc.
  - Values are connected to objects through **attributes** (rather than roles).
- Is equipped with identification assertions.
- Captures most of UML class diagrams and Extended ER diagrams.
- Enjoys nice computational properties, both w.r.t. the traditional reasoning tasks, and w.r.t. query answering (see later).

Outline of Part 4

Syntax of the *DL-Lite*$_{\mathcal{A}}$ description language

- Role expressions:
  - atomic role: P
  - basic role: $Q ::= P \mid P^-$
  - arbitrary role: $R ::= Q \mid \neg Q$ (to express disjointness)
- Concept expressions:
  - atomic concept: A
  - basic concept: $B ::= A \mid \exists Q \mid \delta(U)$
  - arbitrary concept: $C ::= \top_C \mid B \mid \neg B$ (to express disjointness)
- Attribute expressions:
  - atomic attribute: U
  - arbitrary attribute: $V : = U \mid \neg U$ (to express disjointness)
- Value-domain expressions:
  - attribute range: $\rho(U)$
  - RDF datatypes: T_i
  - top domain: $\top_D$

unibz

Semantics of *DL-Lite*$_\mathcal{A}$ – Objects vs. values

	Objects	Values
Interpretation domain $\Delta^\mathcal{I}$	Domain of objects $\Delta_O^\mathcal{I}$	Domain of values $\Delta_V^\mathcal{I}$
Alphabet Γ of constants	Object constants Γ_O	Value constants Γ_V
	$c^\mathcal{I} \in \Delta_O^\mathcal{I}$	$d^\mathcal{I} = \mathit{val}(d)$ given a priori
Unary predicates	Concept C	RDF datatype T_i
	$C^\mathcal{I} \subseteq \Delta_O^\mathcal{I}$	$T_i^\mathcal{I} \subseteq \Delta_V^\mathcal{I}$ given a priori
Binary predicates	Role R	Attribute V
	$R^\mathcal{I} \subseteq \Delta_O^\mathcal{I} \times \Delta_O^\mathcal{I}$	$V^\mathcal{I} \subseteq \Delta_O^\mathcal{I} \times \Delta_V^\mathcal{I}$

unibz

Semantics of the *DL-Lite*$_{\mathcal{A}}$ constructs

Construct	Syntax	Example	Semantics
atomic role	P	child	$P^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}$
inverse role	P^-	child$^-$	$\{(o,o') \mid (o',o) \in P^{\mathcal{I}}\}$
role negation	$\neg Q$	¬manages	$(\Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}) \setminus Q^{\mathcal{I}}$
atomic concept	A	Doctor	$A^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}}$
existential restriction	$\exists Q$	∃child$^-$	$\{o \mid \exists o'.\,(o,o') \in Q^{\mathcal{I}}\}$
concept negation	$\neg B$	¬∃child	$\Delta^{\mathcal{I}} \setminus B^{\mathcal{I}}$
attribute domain	$\delta(U)$	δ(salary)	$\{o \mid \exists v.\,(o,v) \in U^{\mathcal{I}}\}$
top concept	$\top_C$		$\top_C^{\mathcal{I}} = \Delta_O^{\mathcal{I}}$
atomic attribute	U	salary	$U^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}$
attribute negation	$\neg U$	¬salary	$(\Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}) \setminus U^{\mathcal{I}}$
top domain	$\top_D$		$\top_D^{\mathcal{I}} = \Delta_V^{\mathcal{I}}$
datatype	T_i	`xsd:int`	$T_i^{\mathcal{I}} \subseteq \Delta_V^{\mathcal{I}}$ (predefined)
attribute range	$\rho(U)$	ρ(salary)	$\{v \mid \exists o.\,(o,v) \in U^{\mathcal{I}}\}$
object constant	c	`john`	$c^{\mathcal{I}} \in \Delta_O^{\mathcal{I}}$
value constant	d	`'john'`	$\mathit{val}(d) \in \Delta_V^{\mathcal{I}}$ (predefined)

unibz

DL-Lite$_{\mathcal{A}}$ assertions

TBox assertions can have the following forms:

- Inclusion assertions (also called positive inclusions):

$B_1 \sqsubseteq B_2$	concept inclusion	$\rho(U) \sqsubseteq T_i$	value-domain inclusion
$Q_1 \sqsubseteq Q_2$	role inclusion	$U_1 \sqsubseteq U_2$	attribute inclusion

- Disjointness assertions (also called negative inclusions):

$B_1 \sqsubseteq \neg B_2$	concept disjointness		
$Q_1 \sqsubseteq \neg Q_2$	role disjointness	$U_1 \sqsubseteq \neg U_2$	attribute disjointness

- Functionality assertions:

$(\textbf{funct}\ Q)$	role functionality	$(\textbf{funct}\ U)$	attribute functionality

- Identification assertions: $(\textbf{id}\ B\ I_1, \ldots, I_n)$
  where each I_j is a role, an inverse role, or an attribute

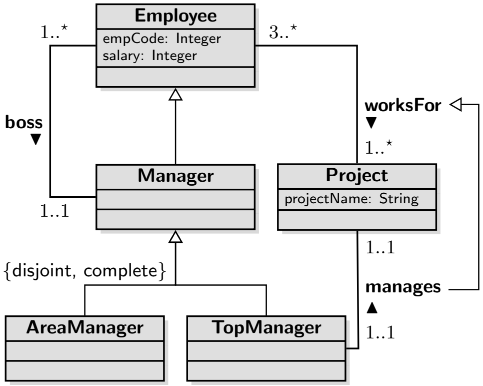
ABox assertions: $A(c)$, $P(c, c')$, $U(c, d)$,
where c, c' are object constants and d is a value constant

Semantics of the *DL-Lite*$_\mathcal{A}$ assertions

Assertion	Syntax	Example	Semantics
conc. incl.	$B_1 \sqsubseteq B_2$	Father $\sqsubseteq$ $\exists$child	$B_1^\mathcal{I} \subseteq B_2^\mathcal{I}$
role incl.	$Q_1 \sqsubseteq Q_2$	father $\sqsubseteq$ anc	$Q_1^\mathcal{I} \subseteq Q_2^\mathcal{I}$
v.dom. incl.	$\rho(U) \sqsubseteq T_i$	ρ(age) $\sqsubseteq$ `xsd:int`	$\rho(U)^\mathcal{I} \subseteq T_i^\mathcal{I}$
attr. incl.	$U_1 \sqsubseteq U_2$	offPhone $\sqsubseteq$ phone	$U_1^\mathcal{I} \subseteq U_2^\mathcal{I}$
conc. disj.	$B_1 \sqsubseteq \neg B_2$	Person $\sqsubseteq$ $\neg$Course	$B_1^\mathcal{I} \subseteq (\neg B_2)^\mathcal{I}$
role disj.	$Q_1 \sqsubseteq \neg Q_2$	sibling $\sqsubseteq$ $\neg$cousin	$Q_1^\mathcal{I} \subseteq (\neg Q_2)^\mathcal{I}$
attr. disj.	$U_1 \sqsubseteq \neg U_2$	offPhn $\sqsubseteq$ $\neg$homePhn	$U_1^\mathcal{I} \subseteq (\neg U_2)^\mathcal{I}$
role funct.	(**funct** Q)	(**funct** father)	$\forall o, o_1, o_2.(o, o_1) \in Q^\mathcal{I} \wedge (o, o_2) \in Q^\mathcal{I} \rightarrow o_1 = o_2$
att. funct.	(**funct** U)	(**funct** ssn)	$\forall o, v, v'.(o, v) \in U^\mathcal{I} \wedge (o, v') \in U^\mathcal{I} \rightarrow v = v'$
id const.	(**id** $B\ I_1, \ldots, I_n$)	(**id** Person name, dob)	$I_1, \ldots, I_n$ identify instances of B
mem. asser.	$A(c)$	Father(bob)	$c^\mathcal{I} \in A^\mathcal{I}$
mem. asser.	$P(c_1, c_2)$	child(bob, ann)	$(c_1^\mathcal{I}, c_2^\mathcal{I}) \in P^\mathcal{I}$
mem. asser.	$U(c, d)$	phone(bob, '2345')	$(c^\mathcal{I}, \mathit{val}(d)) \in U^\mathcal{I}$

unibz

DL-Lite$_\mathcal{A}$ – Example

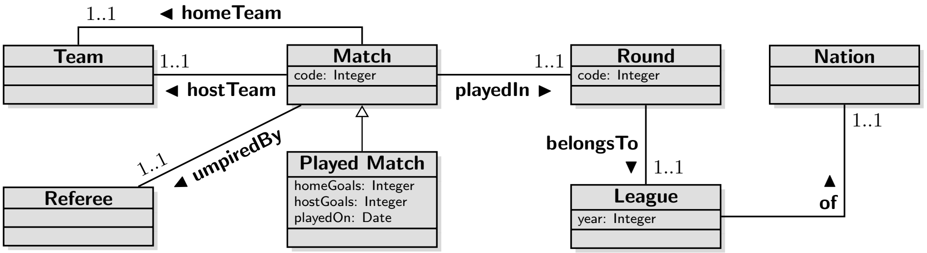


$$
\begin{array}{rcl}
\text{Manager} & \sqsubseteq & \text{Employee}\\
\text{AreaManager} & \sqsubseteq & \text{Manager}\\
\text{TopManager} & \sqsubseteq & \text{Manager}\\
\text{AreaManager} & \sqsubseteq & \neg\text{TopManager}\\
\text{Employee} & \sqsubseteq & \delta(\text{empCode})\\
\delta(\text{empCode}) & \sqsubseteq & \text{Employee}\\
\rho(\text{empCode}) & \sqsubseteq & \texttt{xsd:int}\\
& (\textbf{funct}\ \text{empCode}) & \\
& (\textbf{id}\ \text{Employee empCode}) & \\
\exists\text{worksFor} & \sqsubseteq & \text{Employee}\\
\exists\text{worksFor}^- & \sqsubseteq & \text{Project}\\
\text{Employee} & \sqsubseteq & \exists\text{worksFor}\\
\text{Project} & \sqsubseteq & \exists\text{worksFor}^-\\
& (\textbf{funct}\ \text{manages}) & \\
& (\textbf{funct}\ \text{manages}^-) & \\
\text{manages} & \sqsubseteq & \text{worksFor}\\
& \vdots &
\end{array}
$$

Note: *DL-Lite*$_\mathcal{A}$ cannot capture completeness of a hierarchy. This would require **disjunction** (i.e., **OR**).

Outline of Part 4

Identification assertions – Example



What we would like to additionally capture:

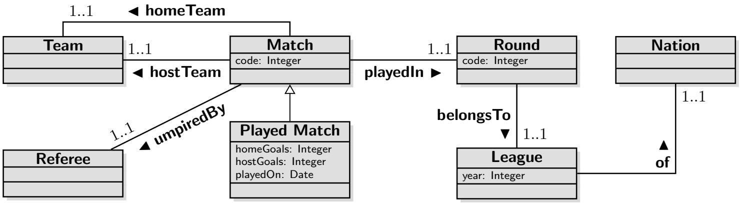
1. No two leagues with the same year and the same nation exist
2. Within a certain league, the code associated to a round is unique
3. Every match is identified by its code within its round
4. Every referee can umpire at most one match in the same round
5. No team can be the home team of more than one match per round
6. No team can be the host team of more than one match per round

Identification assertions – Example (cont'd)

$\mathsf{League} \sqsubseteq \exists\mathsf{of}$	$\mathsf{PlayedMatch} \sqsubseteq \mathsf{Match}$	
$\exists\mathsf{of} \sqsubseteq \mathsf{League}$	$\delta(\mathsf{code}_M) \sqsubseteq \mathsf{Match}$	$\mathsf{Match} \sqsubseteq \delta(\mathsf{code}_M)$
$\exists\mathsf{of}^- \sqsubseteq \mathsf{Nation}$	$\delta(\mathsf{code}_R) \sqsubseteq \mathsf{Round}$	$\mathsf{Round} \sqsubseteq \delta(\mathsf{code}_R)$
$\mathsf{Round} \sqsubseteq \exists\mathsf{belongsTo}$	$\delta(\mathsf{playedOn}_P) \sqsubseteq \mathsf{PlayedMatch}$	$\cdots$
$\exists\mathsf{belongsTo} \sqsubseteq \mathsf{Round}$	$\cdots$	
$\exists\mathsf{belongsTo}^- \sqsubseteq \mathsf{League}$	$\rho(\mathsf{code}_M) \sqsubseteq \mathtt{xsd:int}$	
$\mathsf{Match} \sqsubseteq \exists\mathsf{playedIn}$	$\rho(\mathsf{playedOn}_P) \sqsubseteq \mathtt{xsd:date}$	
$\cdots$	$\cdots$	

(**funct** of)	(**funct** hostTeam)	(**funct** homeGoals)
(**funct** belongsTo)	(**funct** umpiredBy)	(**funct** hostGoals)
(**funct** playedIn)	(**funct** code)	(**funct** playedOn)
(**funct** homeTeam)	(**funct** year)	

Identification assertions – Example (cont'd)



1. No two leagues with the same year and the same nation exist
2. Within a certain league, the code associated to a round is unique
3. Every match is identified by its code within its round
4. Every referee can umpire at most one match in the same round
5. No team can be the home team of more than one match per round
6. No team can be the host team of more than one match per round

(**id** League of, year_L) (**id** Match umpiredBy, playedIn)
(**id** Round belongsTo, code_R) (**id** Match homeTeam, playedIn)
(**id** Match playedIn, code_M) (**id** Match hostTeam, playedIn)

Semantics of identification assertions

Let $(\textbf{id}\ B\ I_1, \ldots, I_n)$ be an identification assertion in a *DL-Lite*$_{\mathcal{A}}$ TBox.

An interpretation $\mathcal{I}$ satisfies such an assertion if for all $o_1, o_2 \in B^{\mathcal{I}}$, if there exist objects or values $u_1, \ldots, u_n$ such that

$$(o_1, u_j) \in I_j^{\mathcal{I}} \text{ and } (o_2, u_j) \in I_j^{\mathcal{I}}, \text{ for } j \in \{1, \ldots, n\},$$

then $o_1 = o_2$.

In other words, the instance o_i of B is identified by the tuple $(u_1, \ldots, u_n)$ of objects or values to which it is connected via $I_1, \ldots, I_n$, respectively.

Note: the roles or attributes I_j are not required to be functional or mandatory.

The above definition of semantics implies that, in the case where an instance $o \in B^{\mathcal{I}}$ is connected by means of $I_j^{\mathcal{I}}$ to a set $u_j^1, \ldots, u_j^k$ of objects (or values), it is each single u_j^h that contributes to the identification of o, and not the whole set $\{u_j^1, \ldots, u_j^k\}$.

Outline of Part 4

Restriction on TBox assertions in *DL-Lite*$_{\mathcal{A}}$ ontologies

We will see that, to ensure the good computational properties that we aim at, we have to impose a **restriction** on the use of functionality and role/attribute inclusions.

Restriction on *DL-Lite*$_{\mathcal{A}}$ TBoxes

No functional or identifying role or attribute can be specialized
by using it in the right-hand side of a role or attribute inclusion assertion.

Formally:

- If $(\textbf{funct } P)$, $(\textbf{funct } P^-)$, $(\textbf{id } B \ \ldots, P, \ldots)$, or $(\textbf{id } B \ \ldots, P^-, \ldots)$ is in $\mathcal{T}$, then $Q \sqsubseteq P$ and $Q \sqsubseteq P^-$ are **not in** $\mathcal{T}$.
- If $(\textbf{funct } U)$ or $(\textbf{id } B \ \ldots, U, \ldots)$ is in $\mathcal{T}$, then $U' \sqsubseteq U$ is **not in** $\mathcal{T}$.

DL-Lite$_\mathcal{F}$ and *DL-Lite*$_\mathcal{R}$

We consider also two sub-languages of *DL-Lite*$_\mathcal{A}$ (that trivially obey the previous restriction):

- *DL-Lite*$_\mathcal{F}$: Allows for functionality assertions, but does not allow for role inclusion assertions.
- *DL-Lite*$_\mathcal{R}$: Allows for role inclusion assertions, but does not allow for functionality assertions.

In both *DL-Lite*$_\mathcal{F}$ and *DL-Lite*$_\mathcal{R}$ we do not consider data values (and hence drop value domains and attributes).

Note: We simply use *DL-Lite* to refer to any of the logics of the *DL-Lite* family.

Outline of Part 4

Capturing basic ontology constructs in *DL-Lite*$_{\mathcal{A}}$

ISA between classes	$A_1 \sqsubseteq A_2$
Disjointness between classes	$A_1 \sqsubseteq \neg A_2$
Mandatory participation to relations	$A_1 \sqsubseteq \exists P \quad A_2 \sqsubseteq \exists P^-$
Domain and range of relations	$\exists P \sqsubseteq A_1 \quad \exists P^- \sqsubseteq A_2$
Functionality of relations	$(\textbf{funct}\ P) \quad (\textbf{funct}\ P^-)$
ISA between relations	$Q_1 \sqsubseteq Q_2$
Disjointness between relations	$Q_1 \sqsubseteq \neg Q_2$
Domain and range of attributes	$\delta(U) \sqsubseteq A \quad \rho(U) \sqsubseteq T_i$
Mandatory and functional attributes	$A \sqsubseteq \delta(U) \quad (\textbf{funct}\ U)$
Identification constraints	$(\textbf{id}\ A\ P, \ldots, P'^-, \ldots, U, \ldots)$

Properties of *DL-Lite*

- The TBox may contain **cyclic dependencies** (which typically increase the computational complexity of reasoning).

  Example: $A \sqsubseteq \exists P, \quad \exists P^- \sqsubseteq A$

- In the syntax, we have not included $\sqcap$ on the right hand-side of inclusion assertions, but it can trivially be added, since

$$B \sqsubseteq C_1 \sqcap C_2 \quad \text{is equivalent to} \quad \begin{array}{lcl} B & \sqsubseteq & C_1 \\ B & \sqsubseteq & C_2 \end{array}$$

- A domain assertion on role P has the form: $\exists P \sqsubseteq A_1$
  A range assertion on role P has the form: $\exists P^- \sqsubseteq A_2$

Properties of *DL-Lite*$_\mathcal{F}$

DL-Lite$_\mathcal{F}$ does **not** enjoy the **finite model property**.

Example

TBox $\mathcal{T}$: $\mathsf{Nat} \sqsubseteq \exists \mathsf{succ}$ $\quad \exists \mathsf{succ}^- \sqsubseteq \mathsf{Nat}$

$\mathsf{Zero} \sqsubseteq \mathsf{Nat} \sqcap \neg \exists \mathsf{succ}^-$ $\quad (\textbf{funct}\ \mathsf{succ}^-)$

ABox $\mathcal{A}$: $\mathsf{Zero}(0)$

$\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ admits only infinite models.
Hence, it is satisfiable, but **not finitely satisfiable**.

Hence, reasoning w.r.t. arbitrary models is different from reasoning w.r.t. finite models only.

Properties of *DL-Lite*$_\mathcal{R}$

- *DL-Lite*$_\mathcal{R}$ **does enjoy the finite model property**. Hence, reasoning w.r.t. finite models is the same as reasoning w.r.t. arbitrary models.
- With role inclusion assertions, we can simulate **qualified existential quantification** in the rhs of an inclusion assertion $A_1 \sqsubseteq \exists Q.A_2$.

  To do so, we introduce a new role Q_{A_2} and:
  - the role inclusion assertion $Q_{A_2} \sqsubseteq Q$
  - the concept inclusion assertions: $$\begin{array}{rcl} A_1 & \sqsubseteq & \exists Q_{A_2} \\ \exists Q_{A_2}^- & \sqsubseteq & A_2 \end{array}$$

  In this way, we can consider $\exists Q.A$ in the right-hand side of an inclusion assertion as an abbreviation.

unibz

Observations on $\mathit{DL\text{-}Lite}_{\mathcal{A}}$

- Captures all the basic constructs of **UML Class Diagrams** and of the **ER Model** . . .
- . . . **except covering constraints** in generalizations.
- Extends (the DL fragment of) the ontology language **RDFS**.
- Is completely symmetric w.r.t. **direct and inverse properties**.
- Is at the basis of the **OWL 2 QL** profile of OWL 2.

The OWL 2 QL Profile

OWL 2 defines three **profiles**: OWL 2 QL, OWL 2 EL, OWL 2 RL.

- Each profile corresponds to a syntactic fragment (i.e., a sub-language) of OWL 2 DL that is targeted towards a specific use.
- The restrictions in each profile guarantee better computational properties than those of OWL 2 DL.

The **OWL 2 QL** profile is derived from the DLs of the *DL-Lite* family:

- "[It] includes most of the main features of conceptual models such as UML class diagrams and ER diagrams."
- "[It] is aimed at applications that use very large volumes of instance data, and where query answering is the most important reasoning task. In OWL 2 QL, conjunctive query answering can be implemented using conventional relational database systems."

Complexity of reasoning in *DL-Lite*$_\mathcal{A}$

1. We have seen that *DL-Lite*$_\mathcal{A}$ can capture the essential features of prominent conceptual modeling formalisms.
2. In the following, we will analyze reasoning in *DL-Lite*, and establish the following characterization of its computational properties:
   - Ontology satisfiability and all classical DL reasoning tasks are:
     - Efficiently tractable in the size of the TBox (i.e., PTIME).
     - Very efficiently tractable in the size of the ABox (i.e., AC^0).
   - Query answering for CQs and UCQs is:
     - PTIME in the size of the TBox.
     - AC^0 in the size of the ABox.
     - Exponential in the size of the **query** (NP-complete).
       Bad? . . . not really, this is exactly as in relational DBs.
3. We will also see that *DL-Lite* is essentially the maximal DL enjoying these nice computational properties.

From (1), (2), and (3) we get that:

DL-Lite is a representation formalism that is very well suited to underlie ontology-based data management systems.

Outline of Part 4

Outline of Part 4

Managing ABoxes

In the traditional DL setting, it is assumed that the data is maintained in the ABox of the ontology:

- 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.
- There may be different ways to manage the ABox from a physical point of view:
  - Description Logics reasoners maintain the ABox is main-memory data structures.
  - When an ABox becomes large, managing it in secondary storage may be required, but this is again handled directly by the reasoner.

Data in external sources

There are several situations where the assumptions of having the data in an ABox managed directly by the ontology system (e.g., a Description Logics reasoner) is not feasible or realistic:

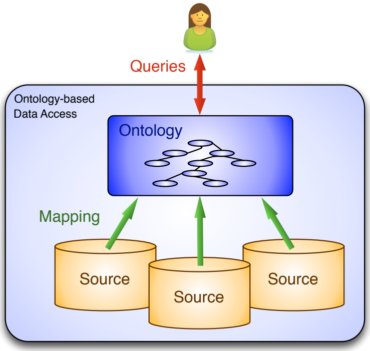
- When the ABox is very large, so that it requires relational database technology.
- When we have no direct control over the data since it belongs to some external organization, which controls the access to it.
- When multiple data sources need to be accessed, such as in Information Integration.

We would like to 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**.

Ontology-based data access: Architecture

The architecture of an OBDA system is based on three main components:

- Ontology: provides a unified, conceptual view of the managed information.
- Data source(s): are external and independent (possibly multiple and heterogeneous).
- Mappings: semantically link data at the sources with the ontology.



The impedance mismatch problem

We have to deal with the **impedance mismatch problem**:

- Sources store data, which is constituted by values taken from concrete domains, such as strings, integers, codes, . . .
- Instead, instances of concepts and relations in an ontology are (abstract) objects.

Solution:

- We need to specify how to construct from the data values in the relational sources the (abstract) objects that populate the ABox of the ontology.
- This specification is embedded in the mappings between the data sources and the ontology.

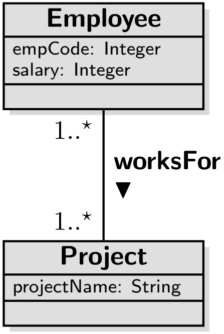
Note: the **ABox** is only **virtual**, and the objects are not materialized.

Solution to the impedance mismatch problem

We need to define a **mapping language** that allows for specifying how to transform data into abstract objects:

- 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).

Impedance mismatch – Example



Actual data is stored in a DB:
– An employee is identified by her SSN.
– A project is identified by its name.

$D_1[\mathit{SSN}\colon \mathrm{String}, \mathit{PrName}\colon \mathrm{String}]$
Employees and projects they work for

$D_2[\mathit{Code}\colon \mathrm{String}, \mathit{Salary}\colon \mathrm{Int}]$
Employee's code with salary

$D_3[\mathit{Code}\colon \mathrm{String}, \mathit{SSN}\colon \mathrm{String}]$
Employee's Code with SSN

...

Intuitively:

- An employee should be created from her SSN: **pers**(*SSN*)
- A project should be created from its name: **proj**(*PrName*)

Creating object identifiers

We need to associate to the data in the tables objects in the ontology.

- We introduce an alphabet Λ of **function symbols**, each with an associated arity.
- To denote values, we use value constants from an alphabet Γ_V.
- To denote objects, we use **object terms** instead of object constants.
  An object term has the form $\mathbf{f}(d_1, \ldots, d_n)$, with $\mathbf{f} \in \Lambda$, and each d_i a value constant in Γ_V.

Example

- If a person is identified by her *SSN*, we can introduce a function symbol $\mathbf{pers}/1$. If `VRD56B25` is a *SSN*, then $\mathbf{pers}(\texttt{VRD56B25})$ denotes a person.
- If a person is identified by her *name* and *dateOfBirth*, we can introduce a function symbol $\mathbf{pers}/2$. Then $\mathbf{pers}(\texttt{Vardi}, \texttt{25/2/56})$ denotes a person.

Mapping assertions

Mapping assertions are used to extract the data from the DB to populate the ontology.

We make use of **variable terms**, which are like object terms, but with variables instead of values as arguments of the functions.

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

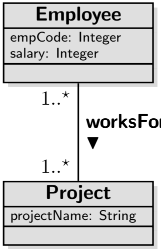
$$\Phi(\vec{x}) \leadsto \Psi(\vec{t}, \vec{y})$$

where

- Φ is an arbitrary SQL query of arity $n > 0$ over $\mathcal{D}$;
- Ψ is a conjunction of atoms whose predicates are atomic concepts and roles of $\mathcal{T}$;
- $\vec{x}$, $\vec{y}$ are variables, with $\vec{y} \subseteq \vec{x}$;
- $\vec{t}$ are variable terms of the form $\mathbf{f}(\vec{z})$, with $\mathbf{f} \in \Lambda$ and $\vec{z} \subseteq \vec{x}$.

unibz

Mapping assertions – Example



D_1[*SSN*: String, *PrName*: String]
Employees and Projects they work for

D_2[*SSN*: String, *Code*: String]
Employee's SSN with code

D_3[*Code*: String, *Salary*: Int]
Employee's code with salary

...

m_1:	`SELECT SSN, PrName`
	`FROM D1`
		$\leadsto$	Employee(**pers**(*SSN*)),
			Project(**proj**(*PrName*)),
			projectName(**proj**(*PrName*), *PrName*),
			worksFor(**pers**(*SSN*), **proj**(*PrName*))

m_2:	`SELECT SSN, Code`
	`FROM D2`
		$\leadsto$	Employee(**pers**(*SSN*)),
			empCode(**pers**(*SSN*), *Code*)

m_3:	`SELECT SSN, Salary`
	`FROM D2, D3`
	`WHERE D2.Code = D3.Code`
		$\leadsto$	Employee(**pers**(*SSN*)),
			salary(**pers**(*SSN*), *Salary*)

Concrete mapping languages

Several proposals for concrete languages to map a relational DB to an ontology:

- They assume that the ontology is populated in terms of RDF triples.
- Some template mechanism is used to specify the triples to instantiate.

Examples: D2RQ[1], SML[2], Ontop[3]

R2RML

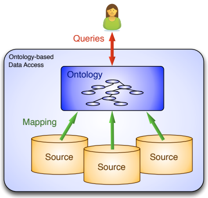
- Most popular RDB to RDF mapping language
- W3C Recommendation 27 Sep. 2012, `http://www.w3.org/TR/r2rml/`
- R2RML mappings are themselves expressed as RDF graphs and written in Turtle syntax.

[1] http://d2rq.org/d2rq-language
[2] http://sparqlify.org/wiki/Sparqlification_mapping_language
[3] https://github.com/ontop/ontop/wiki/ObdalibObdaTurtlesyntax

Outline of Part 4

unibz

Ontology-based data access: Formalization



To formalize OBDA, we distinguish between the intensional and the extensional level information.

Def.: An **OBDA specification** is a triple $\mathcal{P} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$, where:

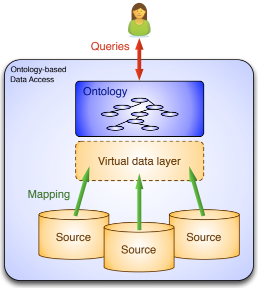
- $\mathcal{T}$ is a DL TBox providing the intensional level of an ontology.
- $\mathcal{S}$ is a (possibly federated) relational database schema for the data sources, possibly with constraints;
- $\mathcal{M}$ is a set of mapping assertions between $\mathcal{T}$ and $\mathcal{S}$.

Def.: An **OBDA system** is a pair $\mathcal{O} = \langle \mathcal{P}, \mathcal{D} \rangle$, where

- $\mathcal{P} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ is an OBDA specification, and
- $\mathcal{D}$ is a relational database compliant with $\mathcal{S}$.

Semantics of an OBDA system: Intuition

In an OBDA system, the **mapping** $\mathcal{M}$ encodes how the data $\mathcal{D}$ in the source(s) $\mathcal{S}$ should be used to populate the elements of the TBox $\mathcal{T}$.



The data $\mathcal{D}$ and the mapping $\mathcal{M}$ define a **virtual data layer** $\mathcal{V}$, which behaves like a (virtual) ABox.

- Queries are answered w.r.t. $\mathcal{T}$ and $\mathcal{V}$.
- One aim is to avoid materializing the data of $\mathcal{V}$.
- Instead, the intensional information in $\mathcal{T}$ and $\mathcal{M}$ is used to translate queries over $\mathcal{T}$ into queries formulated over $\mathcal{S}$.

OBDA vs. Ontology Based Query Answering (OBQA)

OBDA relies on OBQA to process queries w.r.t. the TBox $\mathcal{T}$, but in addition is concerned with efficiently dealing with the mapping $\mathcal{M}$.

OBDA should not be confused with OBQA.

Semantics of mappings

To define the semantics of an OBDA system $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$, we first need to define the semantics of mappings.

Def.: Satisfaction of a mapping assertion with respect to a database

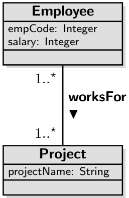
An interpretation $\mathcal{I}$ **satisfies** a mapping assertion $\Phi(\vec{x}) \leadsto \Psi(\vec{t}, \vec{y})$ in $\mathcal{M}$ **with respect to a database** $\mathcal{D}$, if for each tuple of values $\vec{v} \in$ *Eval*$(\Phi, \mathcal{D})$, and for each ground atom in $\Psi[\vec{x}/\vec{v}]$, 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 $(s_1^{\mathcal{I}}, s_2^{\mathcal{I}}) \in P^{\mathcal{I}}$.

Intuitively, $\mathcal{I}$ **satisfies** $\Phi \leadsto \Psi$ w.r.t. $\mathcal{D}$ if all facts obtained by evaluating Φ over $\mathcal{D}$ and then propagating the answers to Ψ, hold in $\mathcal{I}$.

Note: *Eval*$(\Phi, \mathcal{D})$ denotes the result of evaluating Φ over the database $\mathcal{D}$.
$\Psi[\vec{x}/\vec{v}]$ denotes Ψ where each x_i has been substituted with v_i.

Semantics of mappings – Example



D$_1$:

SSN	*PrName*
23AB	tones
...	...

D$_2$:

SSN	*Code*
23AB	e23
...	...

D$_3$:

Code	*Salary*
e23	15000
...	...

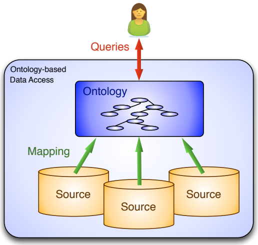
The following interpretation $\mathcal{I}$ satisfies the mapping assertions m_1 and m_3 (we ignore m_2) with respect to the above database:
Note that we have directly used object terms as domain elements.

$$\begin{aligned}\mathcal{I}: \Delta_O^{\mathcal{I}} &= \{\textbf{pers}(\texttt{23AB}), \textbf{proj}(\texttt{tones}), \ldots\}, \quad \Delta_V^{\mathcal{I}} = \{\texttt{tones}, \texttt{15000}, \ldots\}\\ \mathsf{Employee}^{\mathcal{I}} &= \{\textbf{pers}(\texttt{23AB}), \ldots\}, \quad \mathsf{Project}^{\mathcal{I}} = \{\textbf{proj}(\texttt{tones}), \ldots\},\\ \mathsf{projectName}^{\mathcal{I}} &= \{(\textbf{proj}(\texttt{tones}), \texttt{tones}), \ldots\},\\ \mathsf{worksFor}^{\mathcal{I}} &= \{(\textbf{pers}(\texttt{23AB}), \textbf{proj}(\texttt{tones})), \ldots\},\\ \mathsf{salary}^{\mathcal{I}} &= \{(\textbf{pers}(\texttt{23AB}), \texttt{15000}), \ldots\}\end{aligned}$$

m_1:
```
SELECT SSN, PrName
FROM D1
```
$\rightsquigarrow$ Employee(**pers**(*SSN*)),
Project(**proj**(*PrName*)),
projectName(**proj**(*PrName*), *PrName*),
worksFor(**pers**(*SSN*), **proj**(*PrName*))

m_3:
```
SELECT SSN, Salary
FROM D2, D3
WHERE D2.Code = D3.Code
```
$\rightsquigarrow$ Employee(**pers**(*SSN*)),
salary(**pers**(*SSN*), *Salary*)

Semantics of an OBDA system



Model of an OBDA system

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

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

An OBDA system $\mathcal{O}$ is **satisfiable** if it admits at least one model.

Outline of Part 4

Answering queries over an OBDA system

In an OBDA system $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$

- Queries are posed over the TBox $\mathcal{T}$.
- The data needed to answer queries is stored in the database $\mathcal{D}$.
- The mapping $\mathcal{M}$ is used to bridge the gap between $\mathcal{T}$ and $\mathcal{D}$.

Two approaches to exploit the mapping:

- bottom-up approach: simpler, but less efficient
- top-down approach: more sophisticated, but also more efficient

Note: Both approaches require to first **split** the TBox queries in the mapping assertions into their constituent atoms.

Splitting of mappings

A mapping assertion $\Phi \rightsquigarrow \Psi$, where the TBox query Ψ is constituted by the atoms X_1,…,X_k, can be split into several mapping assertions:

$$\Phi \rightsquigarrow X_1 \qquad \cdots \qquad \Phi \rightsquigarrow X_k$$

This is possible, since Ψ does not contain non-distinguished variables.

Example

m_1: `SELECT SSN, PrName FROM D`$_1$ $\rightsquigarrow$ Employee(**pers**(*SSN*)),
Project(**proj**(*PrName*)),
projectName(**proj**(*PrName*), *PrName*),
worksFor(**pers**(*SSN*), **proj**(*PrName*))

is split into

m_1^1: `SELECT SSN, PrName FROM D`$_1$ $\rightsquigarrow$ Employee(**pers**(*SSN*))
m_1^2: `SELECT SSN, PrName FROM D`$_1$ $\rightsquigarrow$ Project(**proj**(*PrName*))
m_1^3: `SELECT SSN, PrName FROM D`$_1$ $\rightsquigarrow$ projectName(**proj**(*PrName*), *PrName*)
m_1^4: `SELECT SSN, PrName FROM D`$_1$ $\rightsquigarrow$ worksFor(**pers**(*SSN*), **proj**(*PrName*))

Bottom-up approach to query answering

Consists in a straightforward application of the mappings:

1. Propagate the data from $\mathcal{D}$ through $\mathcal{M}$, materializing an ABox $\mathcal{A}_{\mathcal{M},\mathcal{D}}$ (the constants in such an ABox are values and object terms).
2. Apply to $\mathcal{A}_{\mathcal{M},\mathcal{D}}$ and to the TBox $\mathcal{T}$, the satisfiability and query answering algorithms developed for *DL-Lite*$_{\mathcal{A}}$.

This approach has several drawbacks (hence is only theoretical):

- The technique is no more AC^0 in the data, since the ABox $\mathcal{A}_{\mathcal{M},\mathcal{D}}$ to materialize is in general polynomial in the size of the data.
- $\mathcal{A}_{\mathcal{M},\mathcal{D}}$ may be very large, and thus it may be infeasible to actually materialize it.
- Freshness of $\mathcal{A}_{\mathcal{M},\mathcal{D}}$ with respect to the underlying data source(s) may be an issue, and one would need to propagate source updates (cf. Data Warehousing).

unibz

Top-down approach to query answering

Consists of three steps:

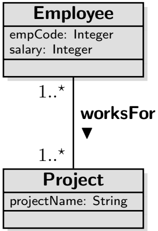
1. **Reformulation:** Compute the perfect reformulation $q_{pr} = \mathit{PerfectRef}(q, \mathcal{T}_P)$ of the original query q, using the inclusion assertions of the TBox $\mathcal{T}$ (see later).
2. **Unfolding:** Compute from q_{pr} a new query q_{unf} by unfolding q_{pr} using (the split version of) the mappings $\mathcal{M}$.
   - Essentially, each atom in q_{pr} that unifies with an atom in Ψ is substituted with the corresponding query Φ over the database.
   - The unfolded query is such that $\mathit{Eval}(q_{unf}, \mathcal{D}) = \mathit{Eval}(q_{pr}, \mathcal{A}_{\mathcal{M},\mathcal{D}})$.
3. **Evaluation:** Delegate the evaluation of q_{unf} to the relational DBMS managing $\mathcal{D}$.

Unfolding

To unfold a query q_{pr} with respect to a set of mapping assertions:

1. For each non-split mapping assertion $\Phi_i(\vec{x}) \rightsquigarrow \Psi_i(\vec{t}, \vec{y})$:
   1. Introduce a **view symbol** Aux_i of arity equal to that of Φ_i.
   2. Add a **view definition** $\mathsf{Aux}_i(\vec{x}) \leftarrow \Phi_i(\vec{x})$.
2. For each split version $\Phi_i(\vec{x}) \rightsquigarrow X_j(\vec{t}, \vec{y})$ of a mapping assertion, introduce a **clause** $X_j(\vec{t}, \vec{y}) \leftarrow \mathsf{Aux}_i(\vec{x})$.
3. Obtain from q_{pr} in all possible ways queries q_{aux} defined over the view symbols Aux_i as follows:
   1. Find a most general unifier ϑ that unifies each atom $X(\vec{z})$ in the body of q_{pr} with the head of a clause $X(\vec{t}, \vec{y}) \leftarrow \mathsf{Aux}_i(\vec{x})$.
   2. Substitute each atom $X(\vec{z})$ with $\vartheta(\mathsf{Aux}_i(\vec{x}))$, i.e., with the body the unified clause to which the unifier ϑ is applied.
4. The unfolded query q_{unf} is the **union** of all queries q_{aux}, together with the view definitions for the predicates Aux_i appearing in q_{aux}.

Unfolding – Example



m_1:
```
SELECT SSN, PrName
FROM D1
```
$\rightsquigarrow$ Employee(**pers**(*SSN*)), Project(**proj**(*PrName*)), projectName(**proj**(*PrName*), *PrName*), worksFor(**pers**(*SSN*), **proj**(*PrName*))

m_2:
```
SELECT SSN, Salary
FROM D2, D3
WHERE D2.Code = D3.Code
```
$\rightsquigarrow$ Employee(**pers**(*SSN*)), salary(**pers**(*SSN*), *Salary*)

We define a view Aux_i for the source query of each mapping m_i.

For each (split) mapping assertion, we introduce a clause:

$$\begin{array}{rcl}
\text{Employee}(\mathbf{pers}(SSN)) & \leftarrow & \text{Aux}_1(SSN, PrName) \\
\text{projectName}(\mathbf{proj}(PrName), PrName) & \leftarrow & \text{Aux}_1(SSN, PrName) \\
\text{Project}(\mathbf{proj}(PrName)) & \leftarrow & \text{Aux}_1(SSN, PrName) \\
\text{worksFor}(\mathbf{pers}(SSN), \mathbf{proj}(PrName)) & \leftarrow & \text{Aux}_1(SSN, PrName) \\
\text{Employee}(\mathbf{pers}(SSN)) & \leftarrow & \text{Aux}_2(SSN, Salary) \\
\text{salary}(\mathbf{pers}(SSN), Salary) & \leftarrow & \text{Aux}_2(SSN, Salary)
\end{array}$$

Unfolding – Example (cont'd)

Query over ontology: employees who work for `tones` and their salary:
$q(e, s) \leftarrow \text{Employee}(e), \text{salary}(e, s), \text{worksFor}(e, p), \text{projectName}(p, \texttt{tones})$

A unifier ϑ between the atoms in q and the clause heads is:

$$\vartheta(e) = \textbf{pers}(\mathit{SSN}) \qquad \vartheta(s) = \mathit{Salary}$$
$$\vartheta(\mathit{PrName}) = \texttt{tones} \qquad \vartheta(p) = \textbf{proj}(\texttt{tones})$$

After applying ϑ to q, we obtain:

$$\begin{aligned} q(\textbf{pers}(\mathit{SSN}), \mathit{Salary}) \leftarrow\ & \text{Employee}(\textbf{pers}(\mathit{SSN})),\ \text{salary}(\textbf{pers}(\mathit{SSN}), \mathit{Salary}),\\ & \text{worksFor}(\textbf{pers}(\mathit{SSN}), \textbf{proj}(\texttt{tones})),\\ & \text{projectName}(\textbf{proj}(\texttt{tones}), \texttt{tones}) \end{aligned}$$

Substituting the atoms with the bodies of the unified clauses, we obtain:

$$\begin{aligned} q(\textbf{pers}(\mathit{SSN}), \mathit{Salary}) \leftarrow\ & \text{Aux}_1(\mathit{SSN}, \texttt{tones}),\ \text{Aux}_2(\mathit{SSN}, \mathit{Salary}),\\ & \text{Aux}_1(\mathit{SSN}, \texttt{tones}),\ \text{Aux}_1(\mathit{SSN}, \texttt{tones}) \end{aligned}$$

Exponential blowup in the unfolding

When there are multiple mapping assertions for each atom, the unfolded query may be exponential in the original one.

Consider a query: $q(y) \leftarrow A_1(y), A_2(y), \ldots, A_n(y)$

and the mappings: $m_i^1\colon \Phi_i^1(x) \rightsquigarrow A_i(\mathbf{f}(x))$ (for $i \in \{1, \ldots, n\}$)
$m_i^2\colon \Phi_i^2(x) \rightsquigarrow A_i(\mathbf{f}(x))$

We add the view definitions: $\mathsf{Aux}_i^j(x) \leftarrow \Phi_i^j(x)$
and introduce the clauses: $A_i(\mathbf{f}(x)) \leftarrow \mathsf{Aux}_i^j(x)$ (for $i \in \{1, \ldots, n\}$, $j \in \{1, 2\}$).

There is a single unifier, namely $\vartheta(y) = \mathbf{f}(x)$, but each atom $A_i(y)$ in the query unifies with the head of two clauses.

Hence, we obtain one unfolded query

$$q(\mathbf{f}(x)) \leftarrow \mathsf{Aux}_1^{j_1}(x), \mathsf{Aux}_2^{j_2}(x), \ldots, \mathsf{Aux}_n^{j_n}(x)$$

for each possible combination of $j_i \in \{1, 2\}$, for $i \in \{1, \ldots, n\}$.
Hence, we obtain 2^n **unfolded queries**.

Computational complexity of query answering

From the top-down approach to query answering, and the complexity results for *DL-Lite*, we obtain the following result.

Theorem

Query answering in a *DL-Lite* OBDM system $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ is

1. NP-**complete** in the size of the query.
2. $\mathrm{PT{\scriptsize IME}}$ in the size of the **TBox** $\mathcal{T}$ and the **mappings** $\mathcal{M}$.
3. AC^0 in the size of the **database** $\mathcal{D}$.

Note: The AC^0 result is a consequence of the fact that query answering in such a setting can be reduced to evaluating an SQL query over the relational database.

Implementation of top-down approach to query answering

To implement the top-down approach, we need to generate an SQL query.

We can follow different strategies:

1. 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.

2. 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.
Experiments have shown that (1) behaves better in most cases.

Towards answering arbitrary SQL queries

- We have seen that answering full SQL (i.e., FOL) queries is undecidable.
- However, we can treat the answers to an UCQ, as "knowledge", and perform further computations on that knowledge.
- This corresponds to applying a knowledge operator to UCQs that are embedded into an arbitrary SQL query (EQL queries) [Calvanese *et al.*, 2007]
    - The UCQs are answered according to the certain answer semantics.
    - The SQL query is evaluated on the facts returned by the UCQs.
- The approach can be implemented by rewriting the UCQs and embedding the rewritten UCQs into SQL.
- The user "sees" arbitrary SQL queries, but these SQL queries are evaluated according to a weakened semantics.

Outline of Part 4

The ONTOP framework

- ONTOP is a framework providing advanced functionalities for representing and reasoning over ontologies of the *DL-Lite* family.
- The basic functionality it offers is query answering of UCQs expressed in SPARQL syntax.
- Query answering is also at the basis of
  - ontology satisfiability;
  - intensional reasoning services: concept/role subsumption and disjunction, concept/role satisfiability.
- Reasoning services are highly optimized.
- Can be used with internal and external DBMS (includes drivers for various commercial and non-commercial DBMSs.
- Implemented in Java as an open source project under the Apache 2 licence.

References I

[Calvanese *et al.*, 2007] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati.

EQL-Lite: Effective first-order query processing in description logics.

In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 274–279, 2007.