Ontology-Based Data Access

Subjects, Issues and Trends

Leopoldo Bertossi

Carleton University

Institute for Data Science

School of Computer Science

Ottawa, Canada

A Start: Metadata in Data Management

- Metadata (MD) is data about data

  An upper layer that gives information about a lower layer

  For example, about the data in relational tables

- We already know about MD in relational DBs: schemas, data types and domains, integrity constraints (ICs)

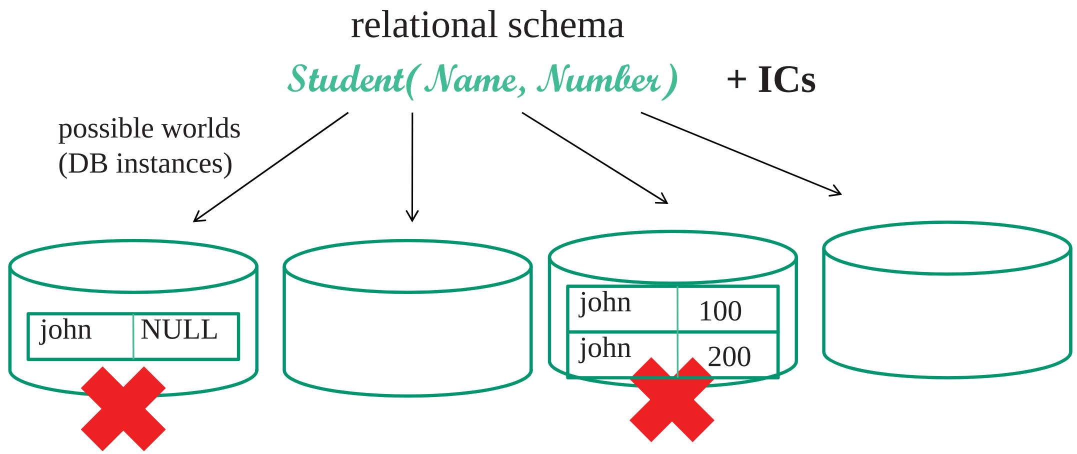
- If ICs are satisfied by the DB (as expected, but not always true), they provide synthetic, higher-level knowledge

  - ICs capture semantics (meaning) of data [3, 11]

  - By filtering out inadmissible (inconsistent) instances, the spectrum of possible instances is narrowed down

    By doing so, better targeting the intended meaning

  - Decreasing uncertainty



+ $\forall xy(Student(x,y) \rightarrow y \neq \text{NULL})$ + $\forall xyz(Student(x,y) \wedge Student(x,z) \rightarrow y = z)$

(capturing semantics via ICs, eliminating possible worlds)

- ICs tell us something about the stored data, still not much though
- ICs can be used, e.g. at query answering time

  For semantic query optimization

- ICs are also useful for interoperability purposes

  When data systems have to interact and possibly be integrated

  They tell us something about what's stored in the data source

- Why not going beyond in terms of MD?

  What else do we know or have after having created a relational DB?

Recovering ER models as Metadata

- When creating a database, we usually start from en entity-relationship (ER) model
- An ER model represents an external, data-related reality

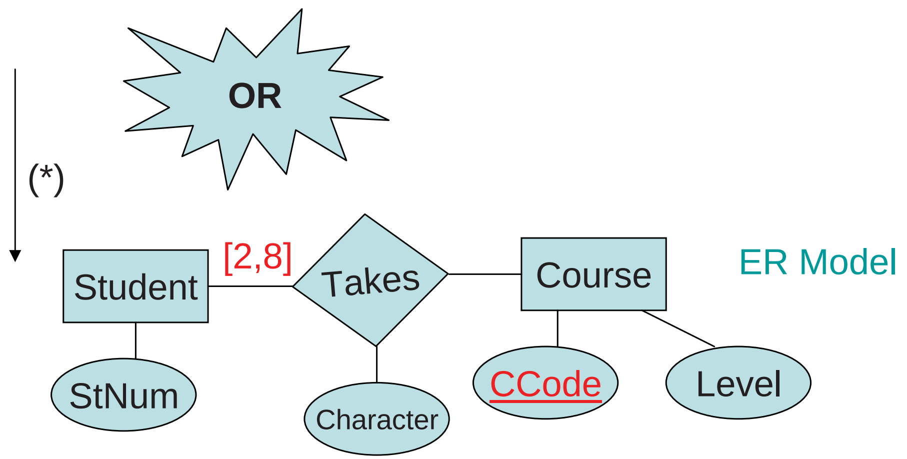
  For example, a model of a business environment

  The model is given as an ER diagram (UML diagram)
- The ER model is closer to the reality than the relational DB to be (which is also a model)
- The ER model is usually forgotten after the DB is created
- The ER model could be used as metadata!

- When creating a relational database, we usually start from an outside reality (OR), e.g. a company, a university, etc.

  We want to model that OR, i.e. produce an abstract, simplified description or representation of OR (leaving aside non-relevant, contingent aspects and details)

- A model can be an ER model, in terms of entities and relationships between them



- For the model to be a good model of OR, it must have a semantics or meaning that corresponds to OR

  ... and keeps the correspondence (*) in place (semantically correct)

- That is why we impose in the model some semantic constraints, like those in red in it

  - A student must take between 2 and 8 courses

  - The course code is a key for the entity: If two objects in *Course* coincide in their values for *CCode*, then their other attribute values must coincide too

- Without those constraints, there could be too many possible ORs that conform to the ER model

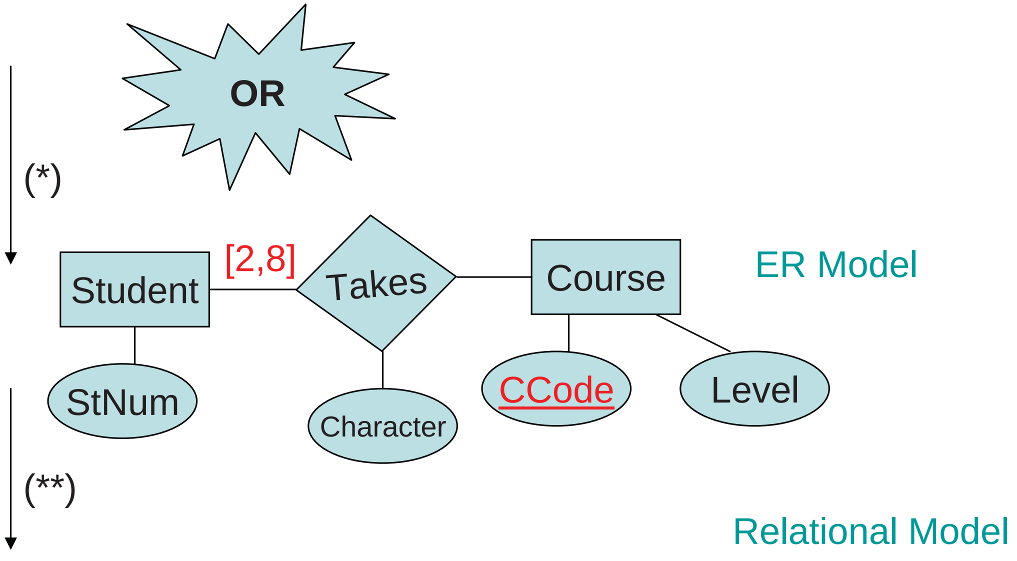
  The model becomes too ambiguous or uncertain

- Imposing semantic constraints eliminates unintended ORs

  ... by narrowing down meaning and filtering out undesirable ORs (other than the intended one)

  We want the ER model to be as close as possible to the initial OR

- The usual next step is producing a relational model from the ER model



- The relational model is also a model of OR

Relational Schema: Student(StNum,Address), Course(<u>CCode</u>,Level,MaxReg), Takes(StNum,CCode,Character) + ICs

- Now a logical model that uses the languages of predicate logic and set theory

- The relational ICs become part of the model, and are also semantic constraints

  Some of them come from the original ER model with its semantics constraints

- As mentioned above, the ER model may be discarded (or not used) after the relational DB is created and populated

  But the ER model contains much semantic information

  It could be put to good use: It could become metadata

  A semantic layer -that can be used with the DB- and is closer to OR and what the user understands

- How to combine a diagrammatic model with a logical model?

  How to realize the integration?

  So that a computer system can take advantage of the combination ...

- The idea is to reconstruct the ER model as a knowledge-base, more precisely, a formal ontology

  Written in some language of symbolic logic (think of something like relational calculus)

- We could borrow languages that have been designed for- or applied to the Semantic Web (SW) initiative [2, 16, 12]

  Some of those languages are being used to express ontologies as metadata for data sources

Interlude on the Semantic Web

- This idea of a semantic layer is at the very basis of the semantic web effort [2, 16]

- The idea is to wrap web sites with descriptions of their contents (resources)

  So that systems that access them will know:

  (a) What to find in them

  What resources, and how they are presented and related

  (b) Conditions satisfied by those resources

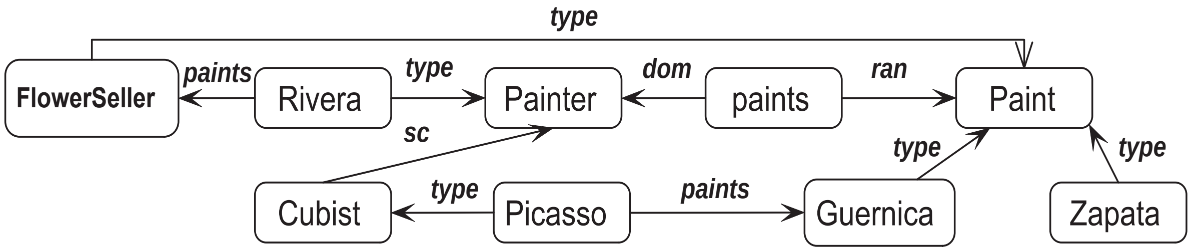
- Useful for querying, integrating and making web sites interoperate

- All this has to be automatized ...

- Logical languages have been created to produce those semantic layers
- Those descriptions become ontologies, which are knowledge-bases expressed in standardized logical languages
- Since all has to be automatized, the ontology languages are expected (not always successfully) to keep a balance between expressive power and difficulty of reasoning
- Languages have been proposed: RDF, RDF-S, OWL (in several versions, light- and heavy-weight), etc.

  E.g. RDF-S has found many applications in data management, and there are multiple RDF-S DBs (check out DBpedia!)
- Some of those languages are being used to express ontologies as metadata for data sources

Example: An RDF-S DB



Here there is data and also conceptual, higher-level, knowledge

In essence, a light-weight ontology

RDF is extended by RDF-S, for “schemas” that can be defined

Together with the data, capturing more semantics

Links ***type*** and ***sc*** (for “subclass), ***dom*** (for “domain), ***ran*** (for “range), have fixed semantics

Properties of a class are inherited by instances that belong to a subclass

When written in logical terms, this semantics has to be specified

ER Models as Ontologies and OBDA

- Logical languages to express metadata can interact with the logical data model (database)

  Being the ER model a diagrammatic model, it can be reconstructed as a symbolic and logic-based ontology

- In general, an ontology is a (logical) description of a set of concepts and their relationships [9]

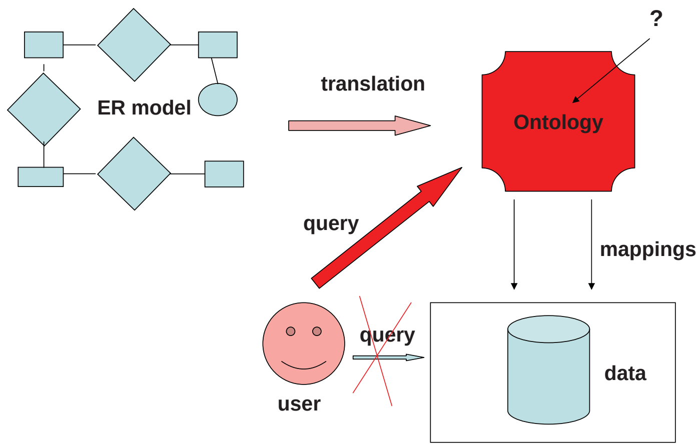
- The ontology becomes metadata, now an explicit and formal ER model

  The ontology (ex ER model) -being closer to the user or business reality- can be used to query the DB

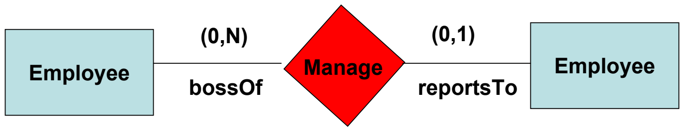
- Querying data sources through ontologies is an active research area

  OBDA: Ontology-based data access [14]

Example: ER model is replaced by (reconstructed as) a symbolic, logical *ontology*



For example, for the following entities/relationship



Introduce basic predicates for the ontology:

- Unary predicates for concepts: $Employee(\cdot)$
- Binary predicates for roles: $BossOf(\cdot,\cdot)$, $ReportsTo(\cdot,\cdot)$

Symbolic statements go into the ontology

E.g. to capture the $(0, 1)$ constraint on the ER's **reportsTo** link:

"Every employee reports to at most one employee" :

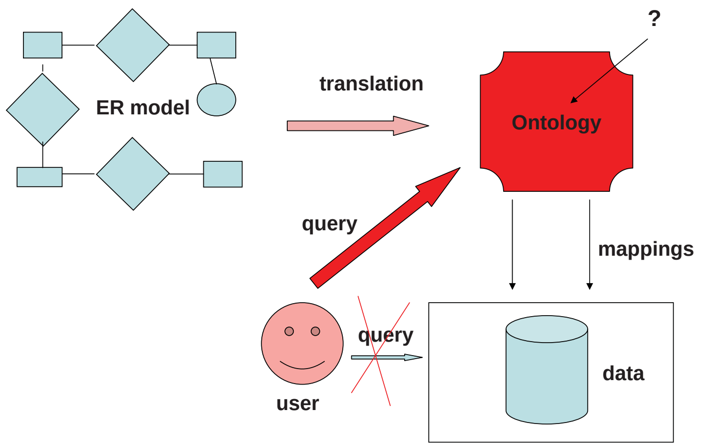
$$\forall x(Employee(x) \rightarrow \exists^{\leq 1} y(Employee(y) \wedge ReportsTo(x, y))$$[1]

A symbolic, machine-processable sentence ...

In OBDA the query language is the language of the ontology

Data stay underneath

Ontology queries are internally "translated" into DB queries

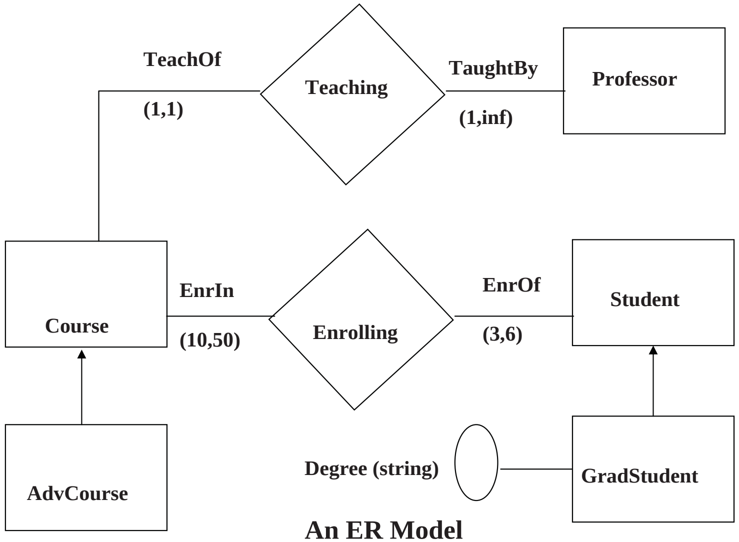


Use mappings between the ontology and the underlying data source(s)

---

[1] I.e., $\forall x(Employee(x) \rightarrow \forall x \forall y_1 y_2((Employee(y_1) \wedge ReportsTo(x, y_1) \wedge Employee(y_2) \wedge ReportsTo(x, y_2)) \rightarrow y_1 = y_2)$.

Example: (for the gist)



An ER Model

$$
\begin{array}{rcl}
\mathit{Teaching} & \sqsubseteq & \forall \mathit{TeachOf}.\mathit{Course} \sqcap \exists^{=1}\mathit{TeachOf} \sqcap \\
 & & \forall \mathit{TaughtBy}.\mathit{Professor} \sqcap \exists^{=1}\mathit{TaughtBy} \\
\mathit{Enrolling} & \sqsubseteq & \forall \mathit{EnrIn}.\mathit{Course} \sqcap \exists^{=1}\mathit{EnrIn} \sqcap \\
 & & \forall \mathit{EnrOf}.\mathit{Student} \sqcap \exists^{=1}\mathit{EnrOf} \\
\mathit{Course} & \sqsubseteq & \forall \mathit{TeachOf}^{-}.\mathit{Teaching} \sqcap \exists^{=1}\mathit{TeachOf}^{-} \sqcap \\
 & & \forall \mathit{EnrIn}^{-}.\mathit{Enrolling} \sqcap \exists^{\geq 10}\mathit{EnrIn}^{-} \sqcap \exists^{\leq 50}\mathit{EnrIn}^{-} \\
\mathit{AdvCourse} & \sqsubseteq & \mathit{Course} \\
\mathit{Professor} & \sqsubseteq & \forall \mathit{TaughtBy}^{-}.\mathit{Teaching} \\
\mathit{Student} & \sqsubseteq & \forall \mathit{EnrOf}^{-}.\mathit{Enrolling} \sqcap \exists^{\geq 3}\mathit{EnrOf}^{-} \sqcap \exists^{\leq 6}\mathit{EnrOf}^{-} \\
\mathit{GradStudent} & \sqsubseteq & \mathit{Student} \sqcap \forall \mathit{Degree}.\mathit{String} \sqcap \exists^{=1}\mathit{Degree}
\end{array}
$$

The link between $\mathbf{AdvCourse}$ and $\mathbf{Course}$ is an IS-A link

As an ontology written in Description Logic (DL)

Entities become DL-concepts

ER links become DL-roles (binary predicates)

ER constraints captured in red in the DL ontology

($\sqsubseteq$ is $\subseteq$ or $\rightarrow$; $\sqcap$ is $\cap$ or $\wedge$; $^{-}$ denotes the inverse role (predicate); original constraints in red)

For illustration, formula at the top of slide 16 could be written in DL as:

$$Employee \sqsubseteq \exists^{\leq 1} Reports.Employee$$

(Here, for a concept C and a role R, the semantics of $\exists^{\leq 1} R.C$ is

$\exists^{\leq 1} R.C = \{x \ : \ |\{y \ : \ R(x, y) \wedge C(y)\}| \leq 1\}$; a form of functional constraint)

- The restricted syntax of DL makes automated reasoning feasible, and sometimes, also efficient

  Notice that full classical predicate logic of which most of the DL variants are fragments is provably undecidable

- By logical reasoning we can infer that constraints that apply to Course also apply to $AdvCourse$

  And less direct logical consequences from the ontology

- In the case of OBDA, the mappings are between unary and binary predicates in the ontology and database predicates (tables), which can be of any arity
- DL is at the basis of SW languages, such as OWL
- The DL ontology above could be written in OWL

Example: A class C defined as the intersection of the classes *Person* and that of the objects all whose children are doctors or have a child who is a doctor

In DL notation:

$Person \sqcap \forall HasChild.(Doctor \sqcup \exists HasChild.Doctor)$

In first-order predicate logic (FOPL) notation (or relational calculus), it defines the class

$\{x \mid Person(x) \wedge \forall HasChild.(Doctor \sqcup \exists HasChild.Doctor)(x)\}$,

i.e. the intersection of two classes; where the second one is

$\{u \mid \forall y(HasChild(u,y) \rightarrow (Doctor(y) \vee \exists z(HasChild(y,z) \wedge Doctor(z)))\}$

Thus, $\forall x(C(x) \equiv (Person(x) \wedge \forall y(HasChild(x,y) \rightarrow$

$$(Doctor(y) \vee \exists z(HasChild(y,z) \wedge Doctor(z)))))$$

In OWL notation (it uses RDF-S syntax): (all the way back to XML, HTML, ...)

$$Person \sqcap \forall HasChild.(Doctor \sqcup \exists HasChild.Doctor)$$

```
<owl:Class>
   <owl:intersectionOf rdf:parseType="collection">
          <owl:Class rdf:about="#Person"/>
          <owl:Restriction>
               <owl:onProperty rdf:resource="#hasChild"/>
               <owl:toClass>
                  <owl:unionOf rdf:parseType="collection">
                     <owl:Class rdf:about="#Doctor"/>
                     <owl:Restriction>
                          <owl:onProperty rdf:resource="#hasChild"/>
                          <owl:hasClass rdf:resource="#Doctor"/>
                     </owl:Restriction>
                  </owl:unionOf>
               </owl:toClass>
          </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>
```

Can be exchanged through and read from web sites, can be used to semantically wrap data, and is machine processable, etc.

In OWL it is possible to express axioms, i.e. general statements about the classes (top table) and properties (bottom table) being used:

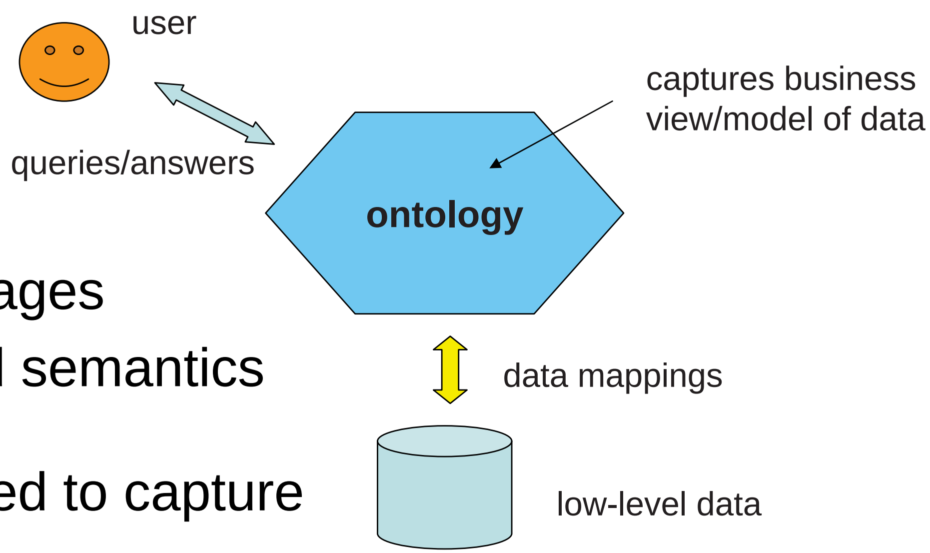
OWL	Example in DL
subClassOf	$Human \sqsubseteq Animal \sqcap Biped$
equivalentClass	$Man \equiv Human \sqcap Male$
disjointWith	$Male \sqsubseteq \neg Female$
sameIndividualAs	$\{President\_Bush\} \equiv \{G\_W\_Bush\}$
differentFrom	$\{john\} \sqsubseteq \neg\{peter\}$
subPropertyOf	$HasDaughter \sqsubseteq HasChild$
equivalentProperty	$Cost \equiv Price$
inverseOf	$HasChild \equiv HasParent^-$
transitiveProperty	$Ancestor^\star \sqsubseteq Ancestor$
functionalProperty	$\top \sqsubseteq \; \leq 1\; HasMother$
inverseFunctionalProperty	$\top \sqsubseteq \; \leq 1\; HasSSN^-$

These axioms are all semantic constraints, about both concepts and properties (roles)

Axioms are not definitions, but basic truths we accept about our domain

- We can see that ontologies can be much more expressive than ER models

- We could start directly with/from an ontology (not necessary coming from an ER model)



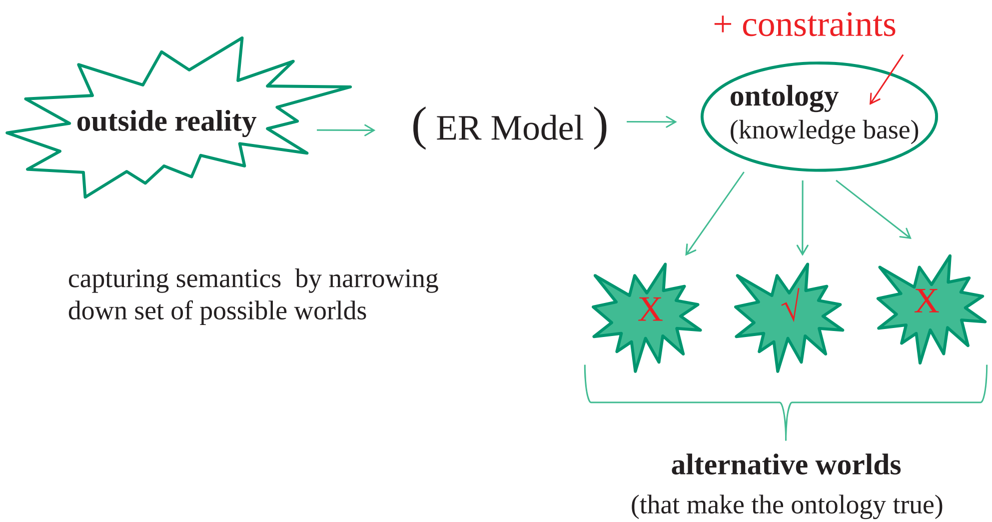
- Their logic-based languages have precise syntax and semantics

- The ontology can be used to capture more semantics

  ... in declarative, precise, and executable terms ...

- It is possible to do automated reasoning from those ontologies

- Via extra logical conditions (constraints) unintended possible worlds that make the initial ontology true can be filtered out (cf. page 3)



- This ontology-based approach enables conceptually simpler and more flexible integration of data management with higher-level reasoning systems

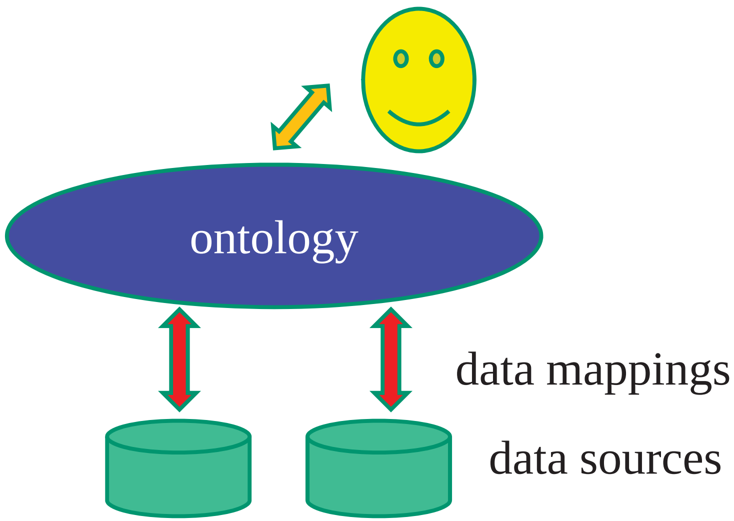
  ... intelligent information systems, knowledge bases, ontologies, semantic web repositories, etc.

- Those ontologies can be useful for interoperability and integration purposes [14, 15]

- Ontologies convey or capture semantics, then sources can be compared in terms of "semantic compatibility" [3, 11]

  Other data sources (at the bottom of the picture above) could be added

  Integrating data sources through/under the same ontology [15]



The Data Integration Connection

- Integrating data sources is a crucial problem in business applications

  Not only there, also for example, in bioinformatics

- Sources can be databases, but also data repositories of all possible kinds

  From structured data (e.g. in relational databases) to documents and WWW pages

- Crucial issue is variety   Data come in all forms, formats, ...

- Heterogeneity is the norm

- There are also semantic issues

- The semantics may be conflicting: think of two mutually contradictory logical theories (ontologies) for two DBs

Example: Two databases with same schema $\mathcal{S} = \{R[A,B], S[B,C]\}$

One DB has the referential IC: $R[B] \subseteq S[B]$
(each value for B in relation R must appear in relation S)

The other has the denial constraint: $not\ (R(A,B), S(B,C))$
(no joins allowed between the two tables)

The two ICs together are inconsistent (in a limited form though: only DBs with empty tables for predicate R can make both true)

- The semantics may be mutually consistent, i.e. their union as logical theories is consistent, but not the combined data

Example: Two student databases, with same schema and same key constraint for the student number

Even if the two DBs separately are consistent, the combination of the two DBs may violate the key constraint in common

Different forms of integration:

- Different basic approaches and paradigms for data integration (DI)

  And hybrid approaches, combinations of the former, that can be combined in complex solutions and systems

  - Materialized: a new physical, material data repository is created

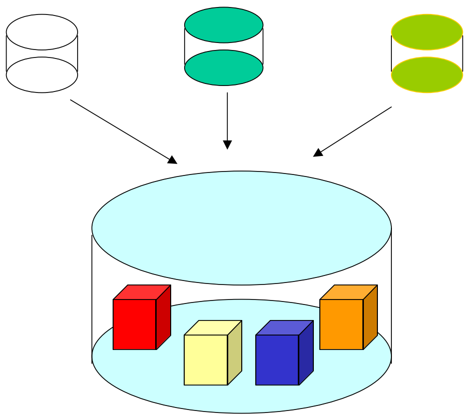
    Best example: Data warehouses (DWHs)

  - Mediated: data stay at the sources, a virtual integration system is created

- In all cases, mappings are needed, to correlate and exchange data between data sources and data targets

Materialized approaches:

A new physical database is created, importing data from other data sources



Data sources may be independent and autonomous

Data warehouses (DWHs): prominent example of materialized DI

Data at the DWH structured differently than those at the sources

Multidimensional business-oriented representation at the DWH

Data cubes in the DWHs, suggesting different dimensions of data

They give context to (usually) numerical data

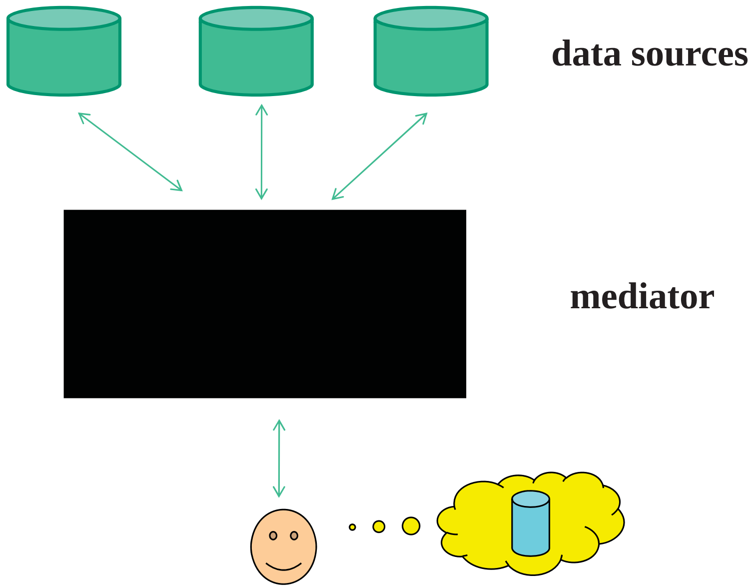
DWH can be conceived as a collection of materialized views, defined on the combination of data sources

Sources and DWHs are meant to be used for different purposes, e.g. transactional/operational vs. business-oriented analysis

Mappings from sources are kept, for refreshment (usually one-directional mappings)

- Virtual data integration (VDI) is via a mediator [17]

  SW system offering DB-like schema interface



- User interacts with mediator

  Data stay at the sources

- Mappings allow to send specific queries to sources and retrieve data

- Notice the similarity with ontology-based data integration (page 26)

Example: OCICS wants to virtually integrate their CU and OU DBs

Sources: Carleton U. Ottawa U.

CUstudents	Number	Name
	101	john
	102	mary

OUstudents	Number	Name
	103	claire
	101	peter

SpecialCU	Number	Field
	101	alg
	102	ai

SpecialOU	Number	Field
	101	db

Single global relation schema, at mediator level

$$Students(Number, Name, Univ, Field)$$

Mapping between the source schemas and the mediated schema?

CUstudents	Number	Name
	101	john
	102	mary

OUstudents	Number	Name
	103	claire
	101	peter

SpecialCU	Number	Field
	101	alg
	102	ai

SpecialOU	Number	Field
	101	db

Mediated schema: $Students(Number, Name, Univ, Field)$

A logical schema mapping: (uses two Datalog rules for view definitions)

$$CUstudents(x, y), SpecialCU(x, z) \rightarrow Students(x, y, \text{‘cu’}, z)$$

$$OUstudents(x, y), SpecialOU(x, z) \rightarrow Students(x, y, \text{‘ou’}, z)$$

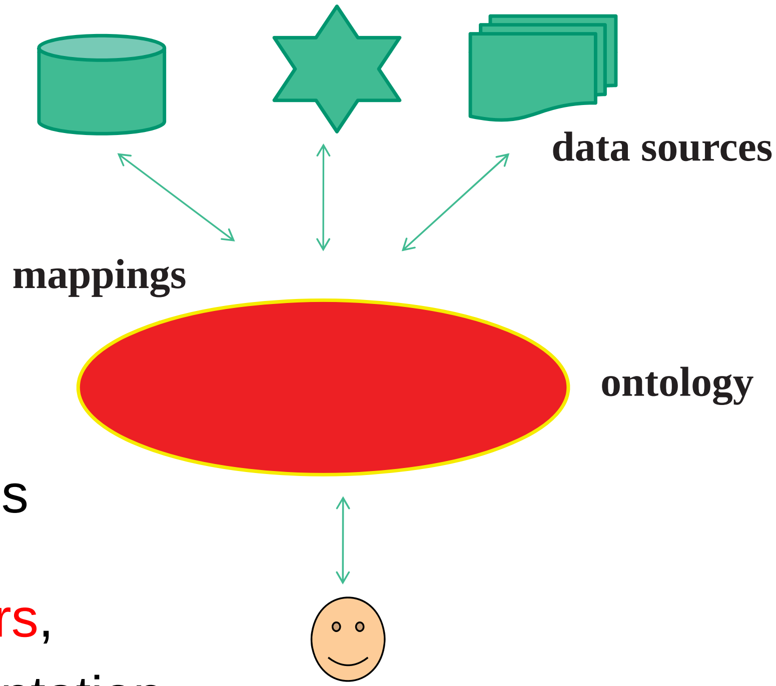
Students becomes a view defined as a disjunction of two conjunctive queries

Global relation as a view of source relations (not the only possibility)

(Can be put as a view defined in relational calculus:

$$\forall xyuz[(CUstudents(x, y) \wedge SpecialCU(x, z) \wedge u = \text{‘cu’}) \vee$$
$$(OUstudents(x, y) \wedge SpecialOU(x, z) \wedge u = \text{‘ou’}) \rightarrow Students(x, y, u, z)]$$

- The mappings above are stored at- and managed by the mediator
- The logical part (the non-procedural components) of the mediator could be conceived as an ontology
- More generally:



Different kinds of sources

Sometimes with wrappers, providing the right presentation for the DI system

What Languages for ODBA?

- DL provides ontological languages (several dialects with different expressive powers)
- Something closer to database practice?
- Datalog has been around for some years in the DB community

As a query and view definition language for relational DBs

As opposed to relational algebra/calculus and older versions of SQL, Datalog provides recursion

$Parent$	A1	A2
	juan	pablo
	adam	cain
	adam	abel
	eve	cain
	pablo	luis
	.	.

$$Ancestor(x, y) \quad \leftarrow \quad Parent(x, y)$$

$$Ancestor(x, z) \quad \leftarrow \quad Ancestor(x, y), Parent(y, z)$$

- Datalog has many nice properties and implementations, but also limited expressive power
- Can we extend Datalog to make it more expressive while keeping most of its nice properties?

Datalog$\pm$ as an Ontological Framework

- Datalog$\pm$ is a family of extensions of classic Datalog

  With new kinds of rules and constraints [6, 8]

- Its languages allow to represent ontological axioms and integrity constraints that cannot be expressed in Datalog

- The idea is to extend Datalog with new constructs to gain expressive power

- While trying to keep the good properties of Datalog:

$\longrightarrow$ declarativity, clear logical semantics, effectiveness & efficiency

(as extensions of whatever is available for Datalog)

Most prominent new ingredients: (the "$+$" in Datalog$\pm$)

- Rules in Datalog$+$ (the extension of Datalog with unrestricted existential rules) admit existentially quantified variables:

$$\exists x P(x,y) \leftarrow R(y,z)$$

  Can be seen as tuple-generating dependencies (TGDs)

- Negative Constraints (NCs): (in particular, denial constraints)

$$\perp \leftarrow P(x,y), R(y,z)$$

- Equality generating dependencies (EGDs):

$$y = z \leftarrow P(x,y), P(x,z)$$

  In this case, a key constraint (KC)

Example: An incomplete EDB D of employers and employees

- Impose on D the TGD (usually as an inclusion dependency):

  "every manager is an employee"

  Expressed by a Datalog rule: $employee(x) \leftarrow manager(x)$

- Another TGD: *"every manager supervises someone"*

  As a rule in Datalog$+$: $\exists y \; supervises(x, y) \leftarrow manager(x)$

- Impose IC: *"employees are not employers"*

  As negative constraint (NC): $\bot \leftarrow employee(x), employer(x)$

- An EGD: *"every employee is supervised by at most one manager"*

$$x = x' \leftarrow supervises(x, y), supervises(x', y)$$

- The "$-$" in Datalog$\pm$ comes from imposing syntactic conditions on Datalog$+$ programs

  For "good" computational behavior

Several applications:

- Express/represent ontologies that interact with data sources
- Represent conceptual data models, and semantic layers on top of databases
- Ontology-Based Data Access (OBDA)
  - Query a database through the ontology
  - In the language of the ontology (closer to the user)
  - Automatically access the underlying data sources
  - Get answers through Datalog evaluation

- Datalog$\pm$ ontologies can represent: ER [7, 5], Semantic Web languages/ontologies [1], UML with object classes [4], ...

  But not classic Datalog!

- Representation of- and navigation in multidimensional data models for data quality assessment and cleaning [13]

Properties & Issues:

- The "$-$" in Datalog$\pm$ refers on syntactic restrictions on Datalog$+$ rules and their (syntactic) interactions
- This limits the gained expressive power
- We can still use Datalog$\pm$ to express ER models and much more
- It can be used as an ontological language
- It can be used as a language to extend incomplete DBs
- The syntactic restrictions ensure that query evaluation (QE) becomes feasible and sometimes efficient

  (Without them, QE under Datalog$\pm$ can be undecidable/non-computable)
- Datalog$\pm$ is still declarative and has a precise and clean semantics

- QE can be implemented
- Probabilistic extensions of Datalog$\pm$ via MLNs

Towards Good Members of the Datalog$\pm$ Family

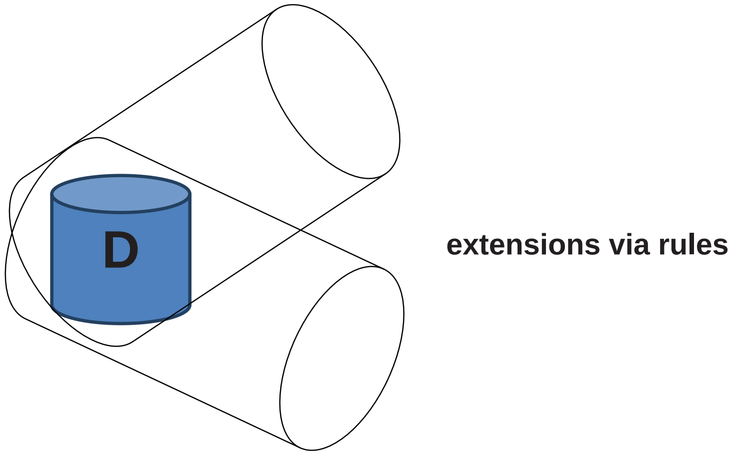
- A Datalog$\pm$ program with a new kind of rules and classical ones is combined with an extensional database (EDB)

- EDB is considered to be incomplete, but extended through the Datalog$\pm$ program

  Generating new tuples for EDB predicates, and full extensions for intensional predicates

- Depending on the kind of rules, possibly several extensions

  Extensions are DBs that extend the EDB and satisfy the rules as classical logical formulas



Whatever is *true in all possible extensions* is considered to be *certain*

- We may want to materialize the extension(s) or keep them virtual

  And query them ... However, ...

- The chase (of the rules on the EDB) generates an instance that extends the EDB and "represents" the whole class of extensions

  It turns out that what is certain is what is true in the chase (i.e. in the extension it produces)

Example: Incomplete EDB $D = \{person(John)\}$

TGDs applied forward (as usual in Datalog), with value invention for existentials

This is the main part of the "chase procedure"

Set Σ of Datalog$+$ rules:

$$\exists x \; father(x, y) \leftarrow person(y)$$
$$person(x) \leftarrow father(x, y)$$

The chase is a procedure that applies the TGDs in a forward manner, generating new tuples

$$\begin{aligned} chase(D, \Sigma) = \{ & father(z_1, John), person(z_1), \\ & father(z_2, z_1), person(z_2), \\ & father(z_3, z_2), person(z_3), \ldots\} \end{aligned}$$

(each z_i is a labeled null value)

- Chase may create non-terminating loops

  So, the chase may not terminate

  Query answering may become undecidable

- Related to (but not necessarily implied by) the fact that ...

  The chase procedure for Datalog$+$ may not terminate, i.e. it produces an infinite extension

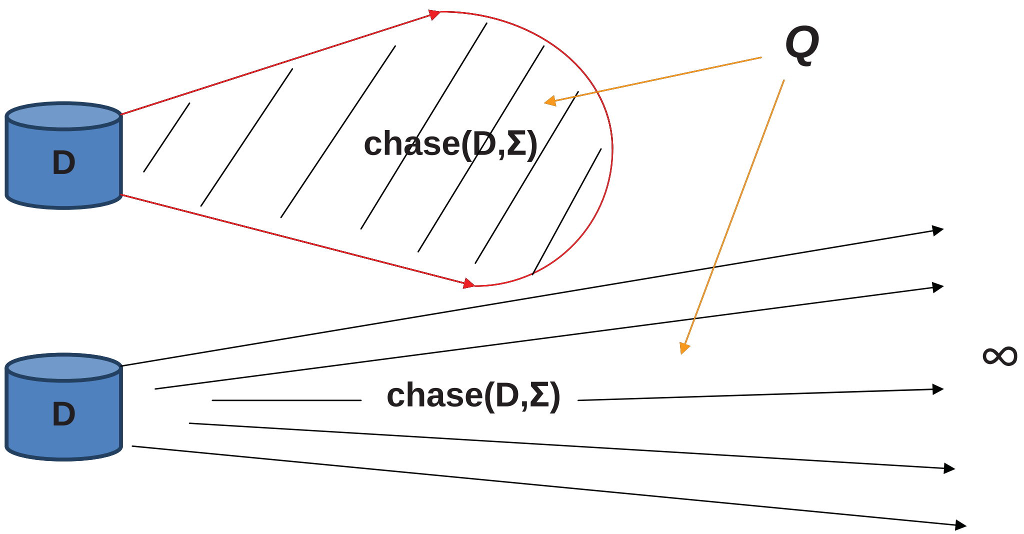
  Finite or infinite, we may still query it ...

- Query answering under Datalog$+$ is indeed undecidable

- Even with infinite chase, things are not always hopeless ...

- Syntactic restrictions of Datalog$\pm$ programs
  - Guarantee decidability of query answering
  - May guarantee efficient query answering ...
- We reserve the name Datalog$\pm$ for the "good" extensions of Datalog

  Each of them can be seen as a syntactic fragment of Datalog$+$



- In first case, QA is obviously decidable

  If the chase can be built in PTIME (in data), QA too

- In second case, QA may be (and sometimes is) undecidable

  But also possibly decidable depending on the program (and the class of queries, but we assume them conjunctive)

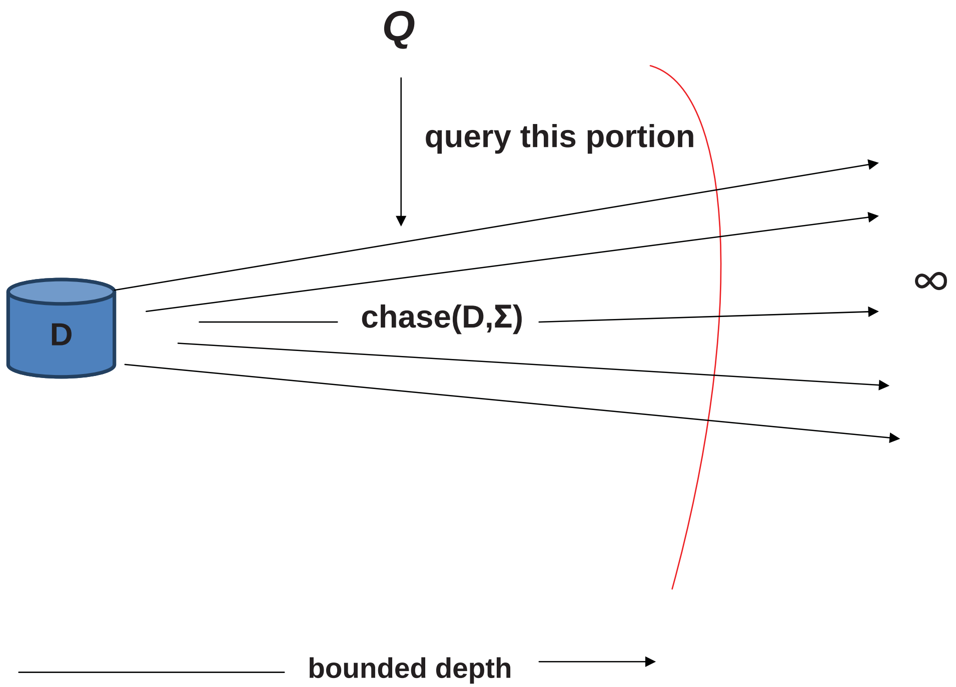
- Good cases of programs that ensure decidability of QA?

  And efficient QA? [10]

- Well-behaved classes of Datalog$\pm$ programs have been considered for the second (infinite) case

- Decidability of QA guaranteed by different syntactic conditions on the set of rules

- The idea is that, depending on the programs, QA can be correctly done by querying only a bounded, initial portion of the chase



Hopefully a "short portion"

Some good classes of Datalog$\pm$ have been identified [5, 6]

Conclusions

- Ontologies have been used for some time in AI (KR) and the Semantic Web

- Now they are being increasingly used in data management

  In particular, in interaction with relational DBs

- Ontologies can be used to access DBs through a model that is close to the user or application environment, e.g. business data

- They can also be used for data integration

- The ontological “schema” can be different from the DB schema

  Connection established via logical mappings

- DL and Datalog$\pm$ have been used for OBDA

- Datalog$\pm$ is a family of extensions of Datalog

  The latter has been around for more than two decades in the DB community

- DL and Datalog$\pm$ have been used to symbolically/logically represent ER, UML, ..., models

- Many applications are still to be unveiled

- There are many interesting open research problems

References

[1] M. Arenas, G. Gottlob, A. Pieris. Expressive Languages for Querying the Semantic Web. Proc. PODS 2014, pp. 14-26.

[2] Tim Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001, pp. 3443.

[3] Alexander Borgida and John Mylopoulos: Data Semantics Revisited. Proc. SWDB, Springer LNCS 3372, 2004, pp. 9-26.

[4] A. Cali, G. Gottlob, G. Orsi and A. Pieris. Querying UML Class Diagrams. Proc. Foundations of Software Science and Computational Structures. Springer LNCS 7213, 2012, pp. 1-25

[5] Andrea Cali, Georg Gottlob and Thomas Lukasiewicz: A General Datalog-Based Framework for Tractable Query Answering over Ontologies. *Journal of Web Semantics*, 2012, 14:57-83.

[6] Andrea Cali, Georg Gottlob and Andreas Pieris. Towards More Expressive Ontology Languages: The Query Answering Problem. *Artificial Intelligence*, 2012, 193:87-128.

[7] Andrea Cali, Georg Gottlob and Andreas Pieris. Ontological Query Answering under Expressive Entity-Relationship Schemata. *Information Systems*, 2012, 37(4):320-335.

[8] Andrea Cali, Georg Gottlob, Thomas Lukasiewicz and Andreas Pieris. A Logical Toolbox for Ontological Reasoning. *SIGMOD Record*, 2011, 40(3):5-14.

[9] B. Chadrasekaran, J. Josephson and V. Richard Benjamins. What are Ontologies, and Why Do We Need Them? IEEE Intelligent Systems, Jan/Feb. 1999, pp. 20-26.

[10] G. Gottlob, G. Orsi and A. Pieris. Query Rewriting and Optimization for Ontological Databases. ACM Trans. Database Syst., 2014, 39(3):25.

[11] David Harel and Bernhard Rumpe. Meaningful Modeling: What's the Semantics of "Semantics"?. *IEEE Computer*, 2004, 37(10): 64-72.

[12] P. Hitzler, M. Krötzsch and S. Rudolph. Foundations of Semantic Web Technologies. CRC Press, 2010.

[13] M. Milani, L. Bertossi and S. Ariyan. Extending Contexts with Ontologies for Multidimensional Data Quality Assessment. Proc. 5th International Workshop on Data Engineering meets the Semantic Web (DESWeb). Data Engineering Workshops (ICDEW), 2014, pp. 242 - 247.

[14] Maurizio Lenzerini. Ontology-Based Data Management. Proc. AMW 2012, CEUR Proceedings, Vol. 866, pp. 12-15.

[15] Alexander Maedche, Boris Motik, Ljiljana Stojanovic, Rudi Studer and Raphael Volz. Ontologies for Enterprise Knowledge Management. *IEEE Intelligent Systems*, 2003, 18(2):26-33.

[16] Nigel Shadbolt, Tim Berners-Lee and Wendy Hall. The Semantic Web Revisited. *IEEE Intelligent Systems*, 2006, 21(3):96-101.

[17] Gio Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 1992, 25(3):38-49.