

Ontologies and Databases

Enrico Franconi

KRDB Research Centre – Free University of Bozen-Bolzano, Italy

<http://www.inf.unibz.it/~franconi>



Summary

- ▶ What is an Ontology
- ▶ (Description Logics for Conceptual Modelling)
- ▶ Queries via a Conceptual Schema

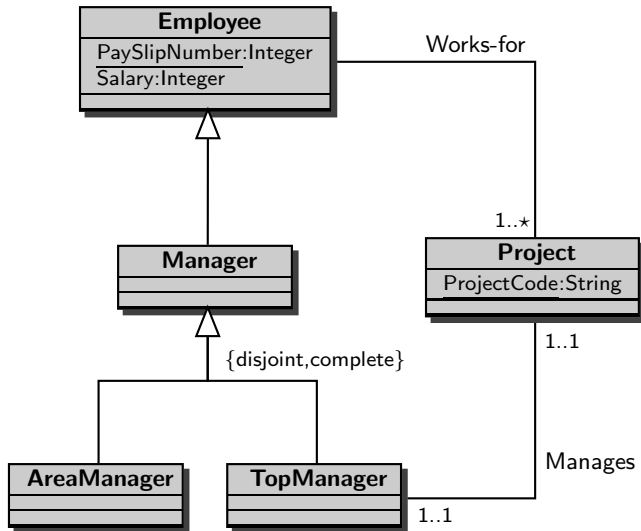
What is an Ontology

- ▶ An ontology is a formal conceptualisation of the world: a **conceptual schema**.
- ▶ An ontology specifies a set of **constraints**, which declare what should necessarily hold in any possible world.
- ▶ Any possible world should conform to the constraints expressed by the ontology.
- ▶ Given an ontology, a *legal world description* is a finite possible world satisfying the constraints.

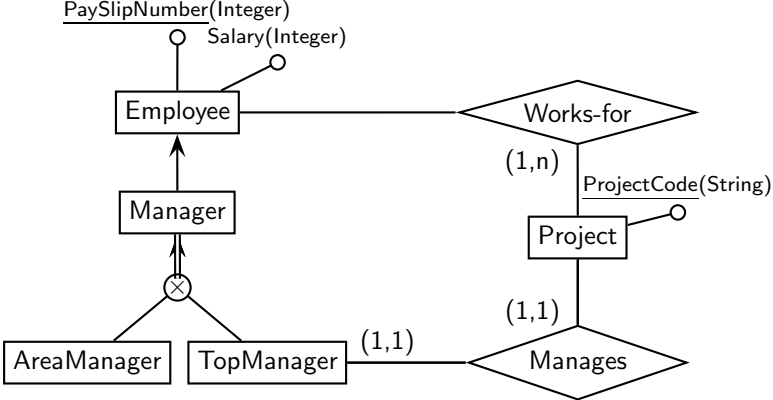
Ontologies and Conceptual Data Models

- ▶ An ontology language usually introduces **concepts** (aka classes, entities), **properties** of concepts (aka slots, attributes, roles), **relationships** between concepts (aka associations), and additional **constraints**.
- ▶ Ontology languages may be simple (e.g., involving only concepts and taxonomies), frame-based (e.g., UML, based on concepts, properties, and binary relationships), or logic-based (e.g. OWL, Description Logics).
- ▶ Ontology languages are typically expressed by means of diagrams.
- ▶ **Entity-Relationship** schemas and **UML** class diagrams can be considered as ontologies.

UML Class Diagram



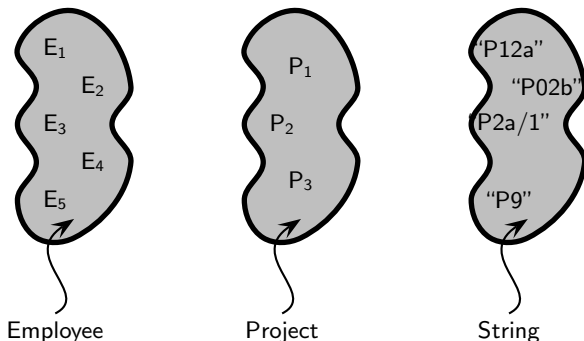
Entity-Relationship Schema



Semantics

In a specific world:

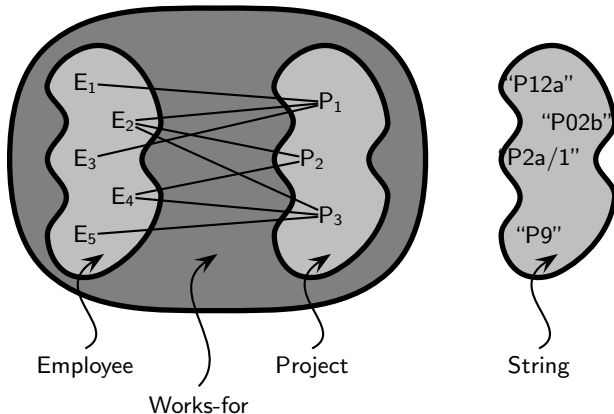
- ▶ A class is a **set of instances**;
- ▶ a n-ary relationship is a **set of n-tuples of instances**;
- ▶ an attribute is a **set of pairs of an instance and a domain element**.



Semantics

In a specific world:

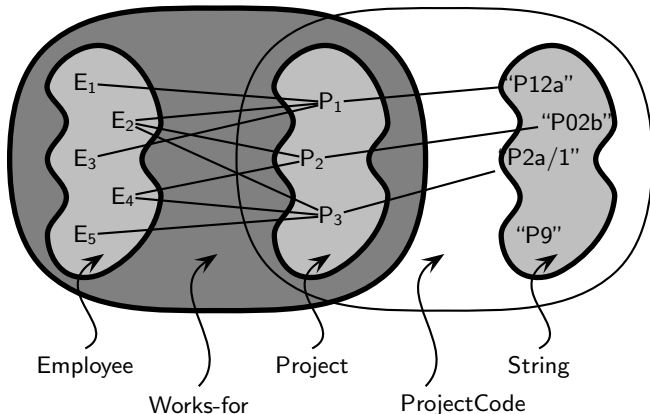
- ▶ A class is a **set of instances**;
- ▶ a n-ary relationship is a **set of n-tuples of instances**;
- ▶ an attribute is a **set of pairs of an instance and a domain element**.



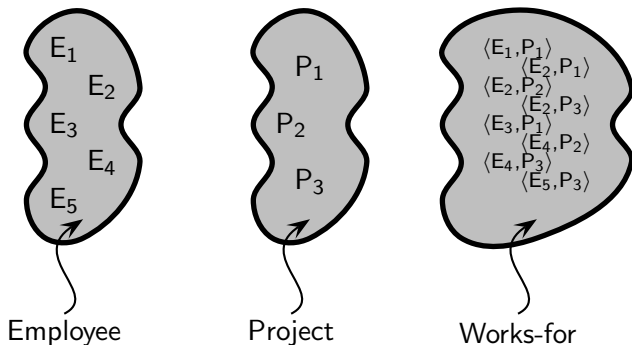
Semantics

In a specific world:

- ▶ A class is a **set of instances**;
- ▶ a n-ary relationship is a **set of n-tuples of instances**;
- ▶ an attribute is a **set of pairs of an instance and a domain element**.



A world is described by sets of instances



The relational representation of a world

Employee

<i>employeeId</i>
E ₁
E ₂
E ₃
E ₄
E ₅

Project

<i>projectId</i>
P ₁
P ₂
P ₃

String

<i>anystring</i>
"P12a"
"P02b"
"P2a/1"
"P9"
...

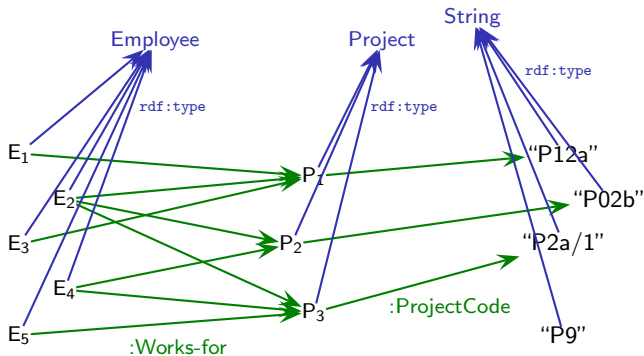
Works-for

<i>employeeId</i>	<i>projectId</i>
E ₁	P ₁
E ₂	P ₁
E ₂	P ₂
E ₂	P ₃
E ₃	P ₁
E ₄	P ₂
E ₄	P ₃
E ₅	P ₃

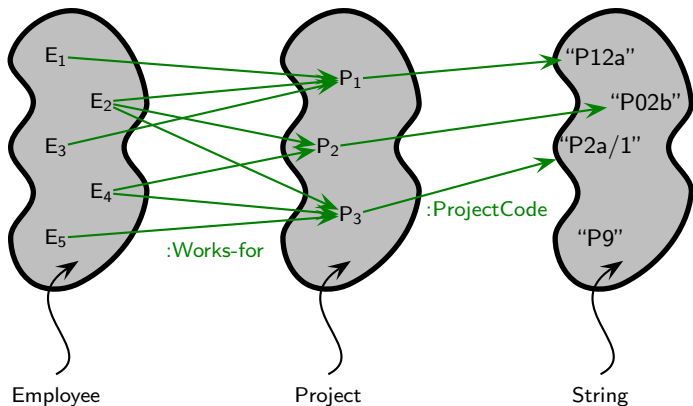
ProjectCode

<i>projectId</i>	<i>pcode</i>
P ₁	"P12a"
P ₂	"P02b"
P ₃	"P2a/1"

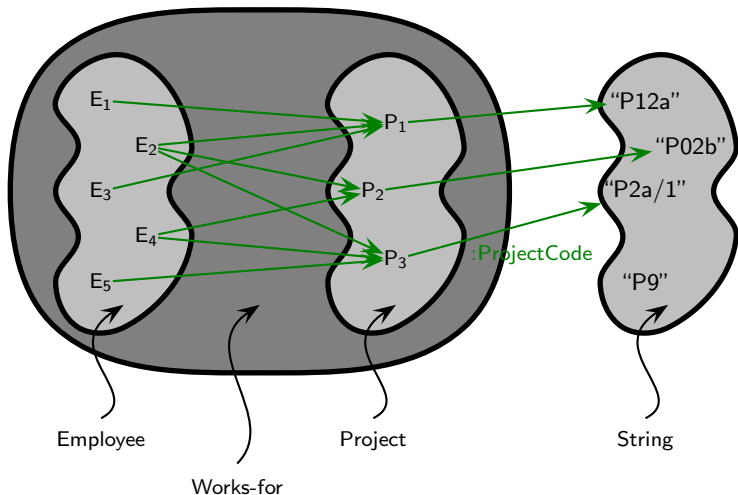
The graph representation of a world – e.g. RDF triples



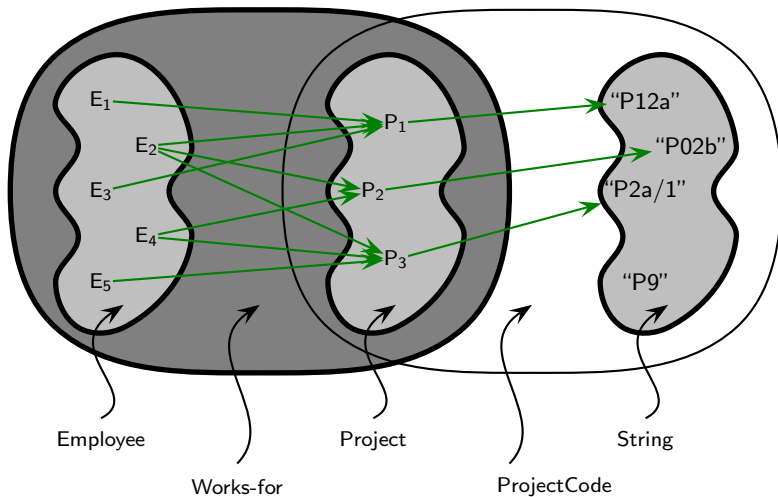
The graph representation of a world – e.g. RDF triples



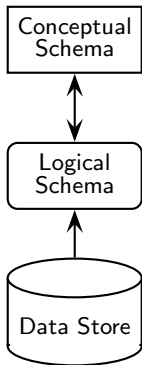
The graph representation of a world – e.g. RDF triples



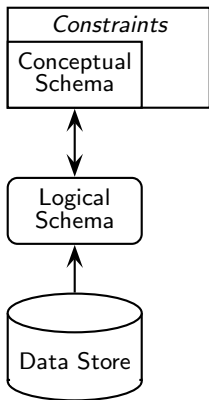
The graph representation of a world – e.g. RDF triples



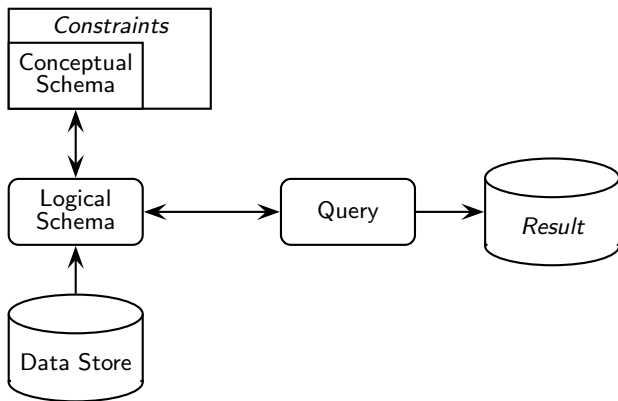
The role of a Conceptual Schema



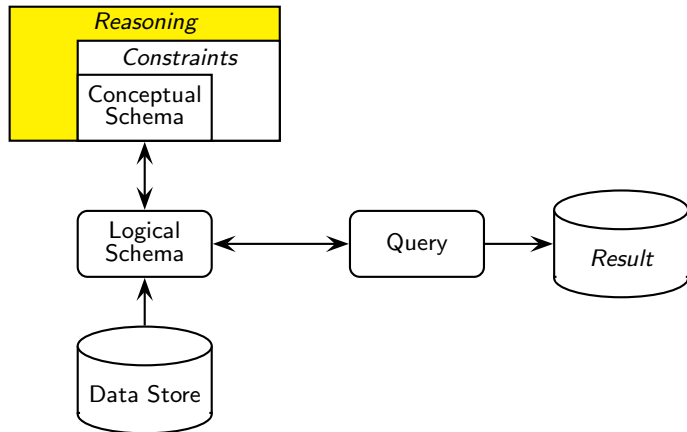
The role of a Conceptual Schema



The role of a Conceptual Schema



The role of a Conceptual Schema

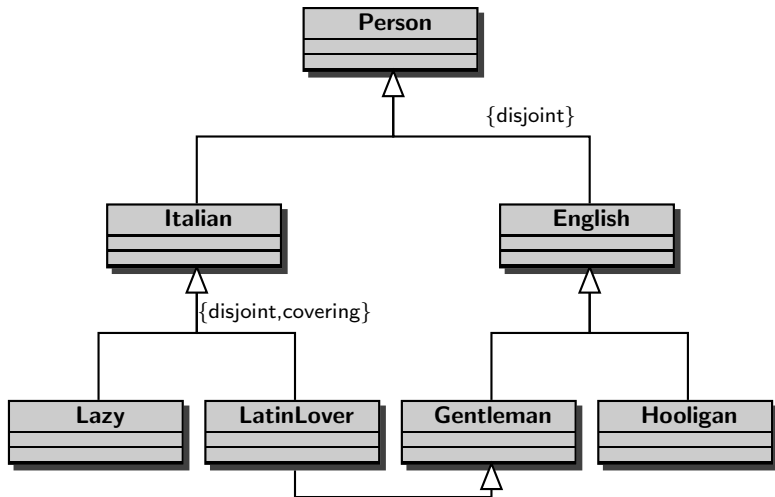


Reasoning

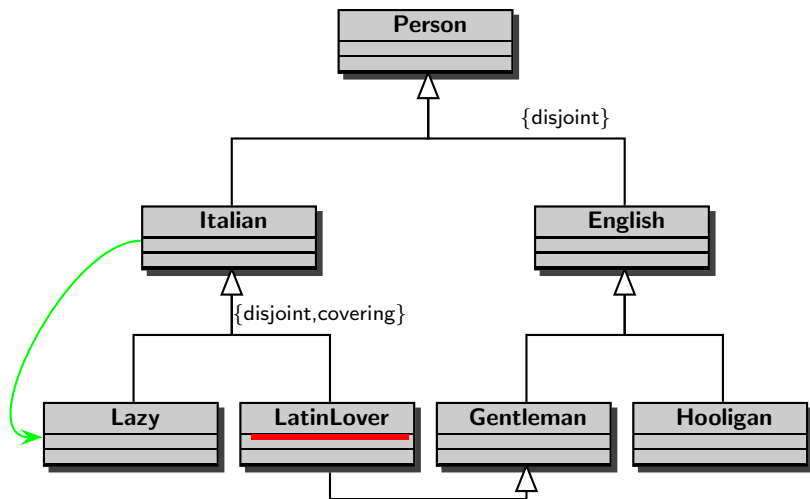
Given an ontology – seen as a collection of constraints – it is possible that additional constraints can be inferred.

- ▶ A class is **inconsistent** if it denotes the empty set in any legal world description.
- ▶ A class is a **subclass** of another class if the former denotes a subset of the set denoted by the latter in any legal world description.
- ▶ Two classes are **equivalent** if they denote the same set in any legal world description.
- ▶ A **stricter** constraint is inferred – e.g., a **cardinality** constraint – if it holds in in any legal world description.
- ▶ ...

Simple reasoning example



Simple reasoning example

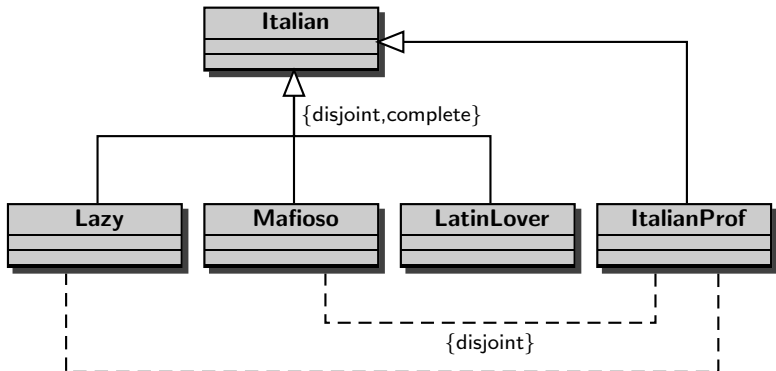


LatinLover = \emptyset

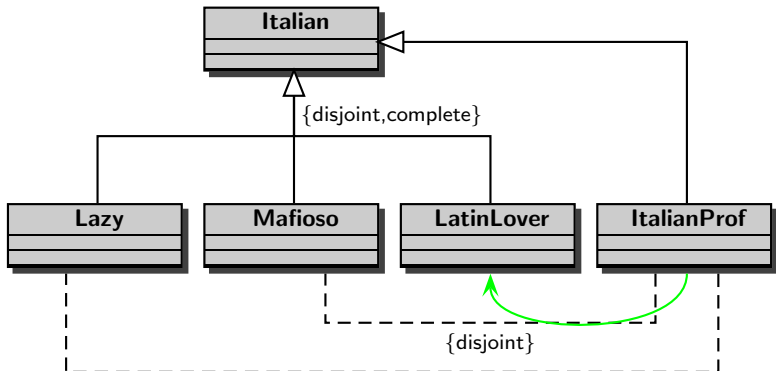
Italian \subseteq Lazy

Italian \equiv Lazy

Reasoning: cute professors



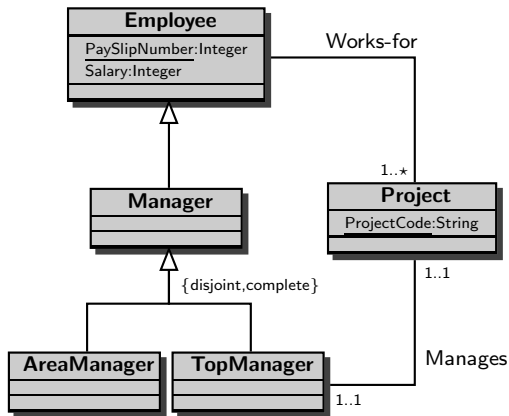
Reasoning: cute professors



implies

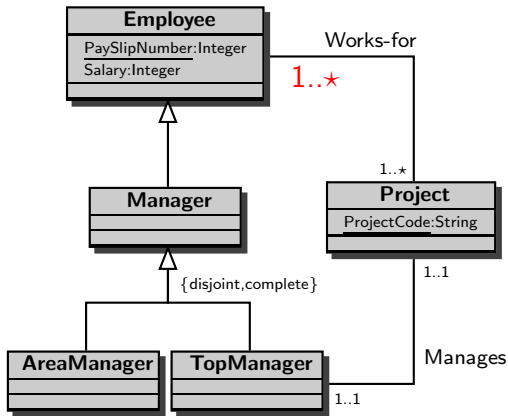
$\text{ItalianProf} \subseteq \text{LatinLover}$

Reasoning with Conceptual Schemas



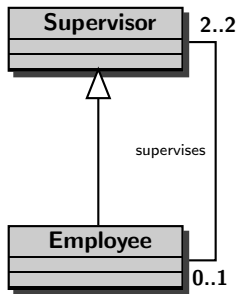
- Managers do not work for a project (she/he just manages it):
 $\forall x. Manager(x) \rightarrow \forall y. \neg WORKS-FOR(x, y)$

Reasoning with Conceptual Schemas

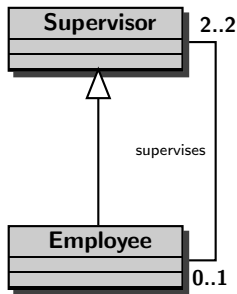


- ▶ Managers do not work for a project (she/he just manages it):
 $\forall x. \text{Manager}(x) \rightarrow \forall y. \neg \text{WORKS-FOR}(x, y)$
- ▶ If the **minimum cardinality** for the participation of employees to the *works-for* relationship is increased, then ...

The democratic company



The democratic company

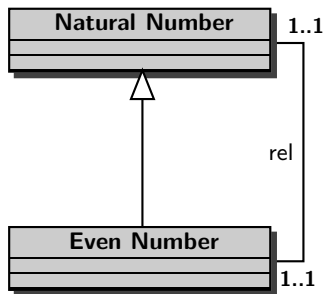


implies

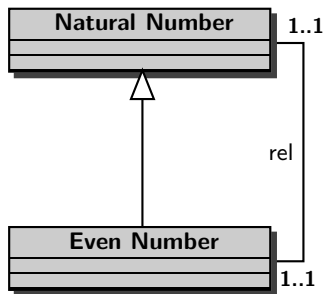
“the classes **Employee** and **Supervisor** necessarily contain an infinite number of instances”.

Since legal world descriptions are *finite* possible worlds satisfying the constraints imposed by the conceptual schema, **the schema is inconsistent**.

How many numbers?



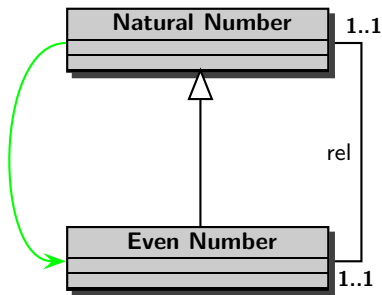
How many numbers?



implies

“the classes **Natural Number** and **Even Number** contain the same number of instances”.

How many numbers?



implies

“the classes **Natural Number** and **Even Number** contain the same number of instances”.

Only if the domain is finite: **Natural Number** \equiv **Even Number**

Summary

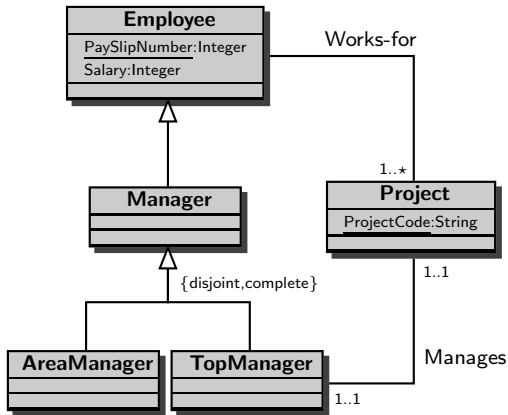
- ▶ Logic and Conceptual Modelling
- ▶ Description Logics for Conceptual Modelling
- ▶ Queries via a Conceptual Schema

Encoding Conceptual Schemas in (Description) Logics

- ▶ Object-oriented data models (e.g., UML and ODMG)
- ▶ Semantic data models (e.g., EER and ORM)
- ▶ Frame-based and web ontology languages (e.g., RDFS, OWL)

Encoding Conceptual Schemas in (Description) Logics

- ▶ Object-oriented data models (e.g., UML and ODMG)
- ▶ Semantic data models (e.g., EER and ORM)
- ▶ Frame-based and web ontology languages (e.g., RDFS, OWL)
- ▶ Theorems **prove** that a conceptual schema and its encoding as DL knowledge bases constrain every world description in the same way – i.e., the models of the DL theory correspond to the legal world descriptions of the conceptual schema, and vice-versa.



Works-for	\sqsubseteq	$\text{emp}/2 : \text{Employee} \sqcap \text{act}/2 : \text{Project}$
Manages	\sqsubseteq	$\text{man}/2 : \text{TopManager} \sqcap \text{prj}/2 : \text{Project}$
Employee	\sqsubseteq	$\exists^{=1}[\text{worker}](\text{PaySlipNumber} \sqcap \text{num}/2 : \text{Integer}) \sqcap$ $\exists^{=1}[\text{payee}](\text{Salary} \sqcap \text{amount}/2 : \text{Integer})$
T	\sqsubseteq	$\exists^{\leq 1}[\text{num}](\text{PaySlipNumber} \sqcap \text{worker}/2 : \text{Employee})$
Manager	\sqsubseteq	$\text{Employee} \sqcap (\text{AreaManager} \sqcup \text{TopManager})$
AreaManager	\sqsubseteq	$\text{Manager} \sqcap \neg \text{TopManager}$
TopManager	\sqsubseteq	$\text{Manager} \sqcap \exists^{=1}[\text{man}]\text{Manages}$
Project	\sqsubseteq	$\exists^{\geq 1}[\text{act}]\text{Works-for} \sqcap \exists^{=1}[\text{prj}]\text{Manages}$
...		

Set-based Constraints

Works-for \subseteq Employee \times Project

Manages \subseteq TopManager \times Project

Employee $\subseteq \{e \mid \#(\text{PaySlipNumber} \cap (\{e\} \times \text{Integer})) \geq 1\}$

Employee $\subseteq \{e \mid \#(\text{Salary} \cap (\{e\} \times \text{Integer})) \geq 1\}$

Project $\subseteq \{p \mid \#(\text{ProjectCode} \cap (\{p\} \times \text{String})) \geq 1\}$

TopManager $\subseteq \{m \mid 1 \geq \#(\text{Manages} \cap (\{m\} \times \Omega)) \geq 1\}$

Project $\subseteq \{p \mid 1 \geq \#(\text{Manages} \cap (\Omega \times \{p\})) \geq 1\}$

Project $\subseteq \{p \mid \#(\text{Works-for} \cap (\Omega \times \{p\})) \geq 1\}$

Manager \subseteq Employee

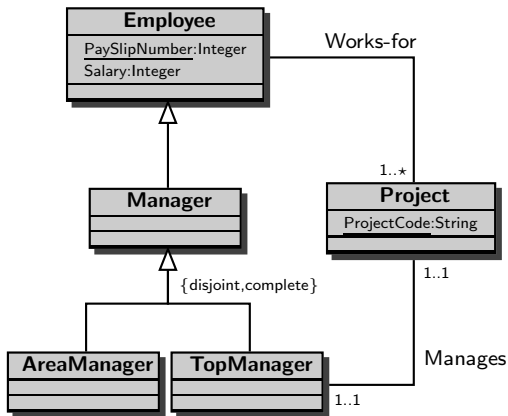
AreaManager \subseteq Manager

TopManager \subseteq Manager

AreaManager \cap TopManager = \emptyset

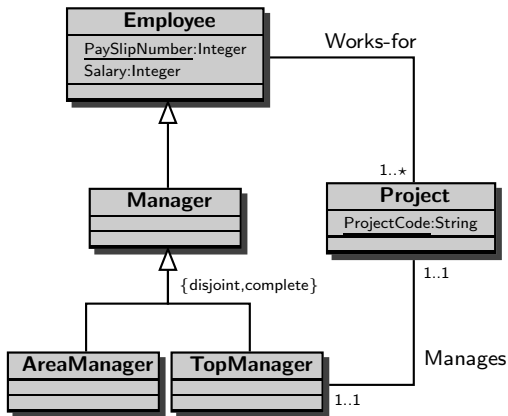
Manager \subseteq AreaManager \cup TopManager

Deducing constraints



Managers are employees who do not work for a project (she/he just manages it):
 $\text{Employee} \sqcap \neg(\exists^{\geq 1}[\text{emp}]\text{Works-for}) \sqsubseteq \text{Manager}, \quad \text{Manager} \sqsubseteq \neg(\exists^{\geq 1}[\text{emp}]\text{Works-for})$

Deducing constraints



Managers are employees who do not work for a project (she/he just manages it):
 $\text{Employee} \sqcap \neg(\exists^{\geq 1}[\text{emp}]\text{Works-for}) \sqsubseteq \text{Manager}, \quad \text{Manager} \sqsubseteq \neg(\exists^{\geq 1}[\text{emp}]\text{Works-for})$

⊨ For every project, there is at least one employee who is not a manager:
 $\text{Project} \sqsubseteq \exists^{\geq 1}[\text{act}](\text{Works-for} \sqcap \text{emp} : \neg\text{Manager})$

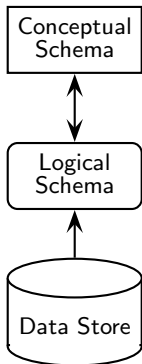
i●com: Intelligent Conceptual Modelling

- ▶ i●com allows for the specification of multiple EER (or UML) diagrams and inter- and intra-schema constraints;
- ▶ Complete logical reasoning is employed by the tool using a hidden underlying DLR inference engine;
- ▶ i●com verifies the specification, infers implicit facts and stricter constraints, and manifests any inconsistencies during the conceptual modelling phase.

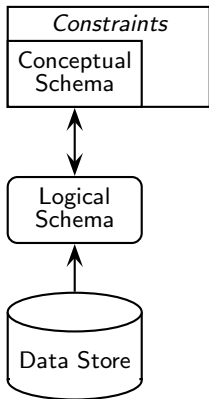
Summary

- ▶ Logic and Conceptual Modelling
- ▶ Description Logics for Conceptual Modelling
- ▶ **Queries via a Conceptual Schema**

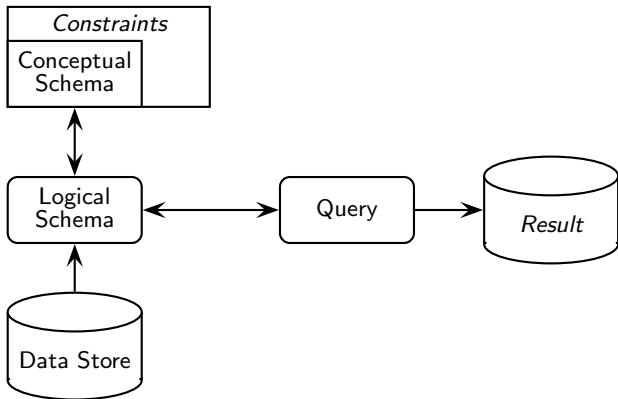
The role of a Conceptual Schema – revisited



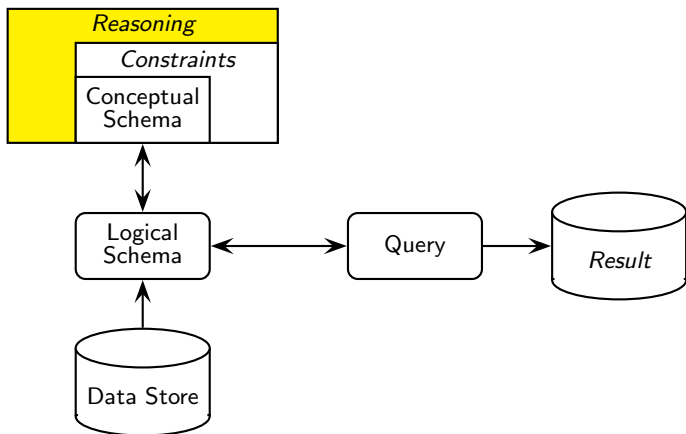
The role of a Conceptual Schema – revisited



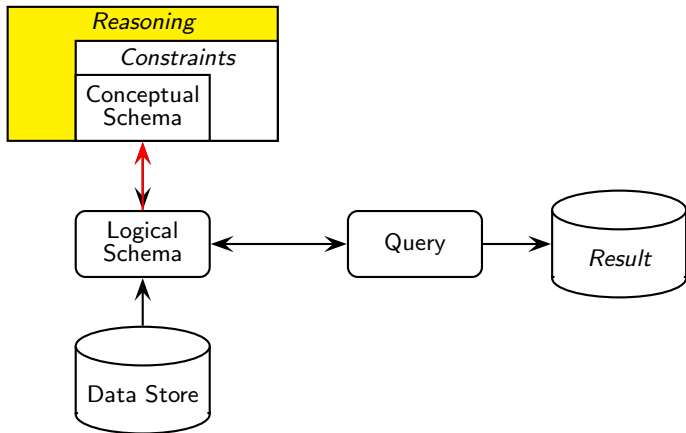
The role of a Conceptual Schema – revisited



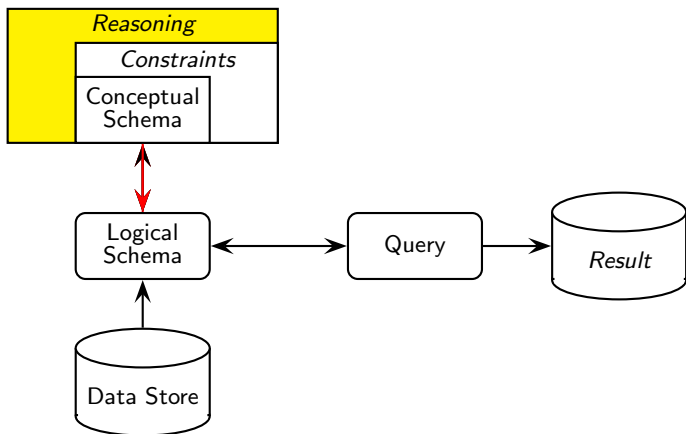
The role of a Conceptual Schema – revisited



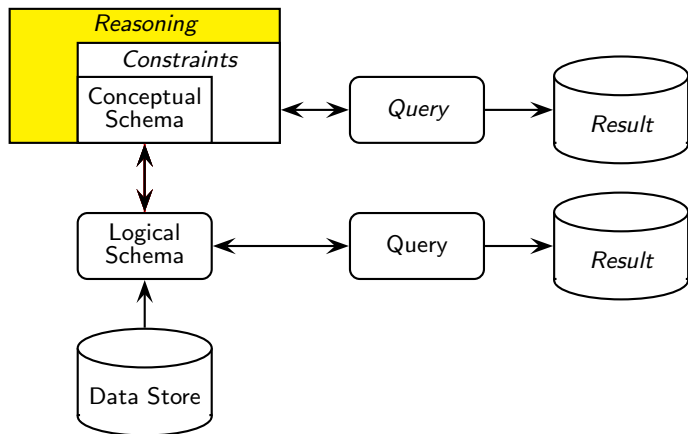
The role of a Conceptual Schema – revisited



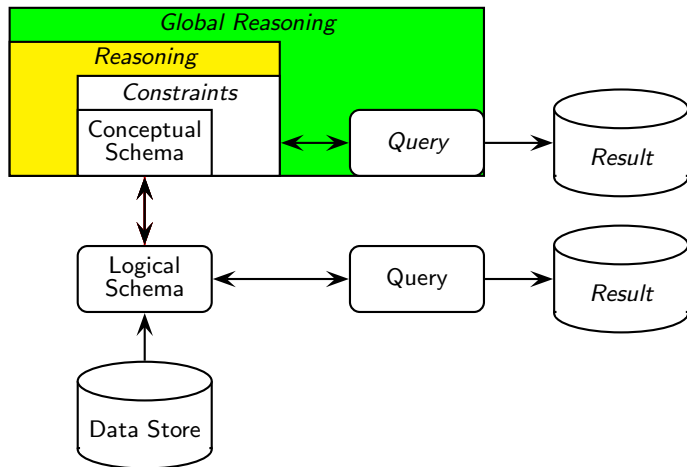
The role of a Conceptual Schema – revisited



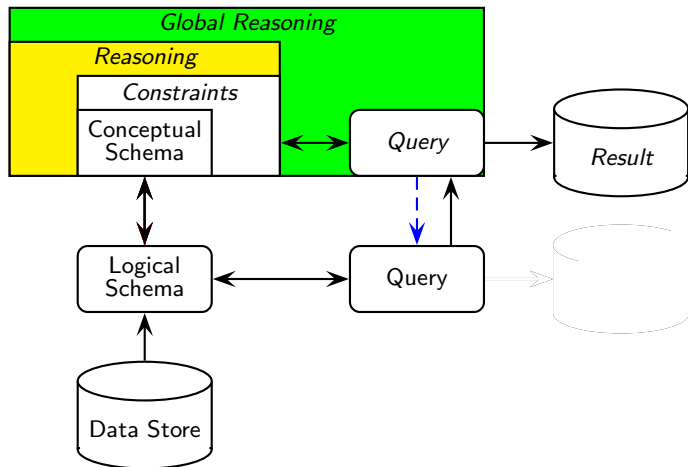
The role of a Conceptual Schema – revisited



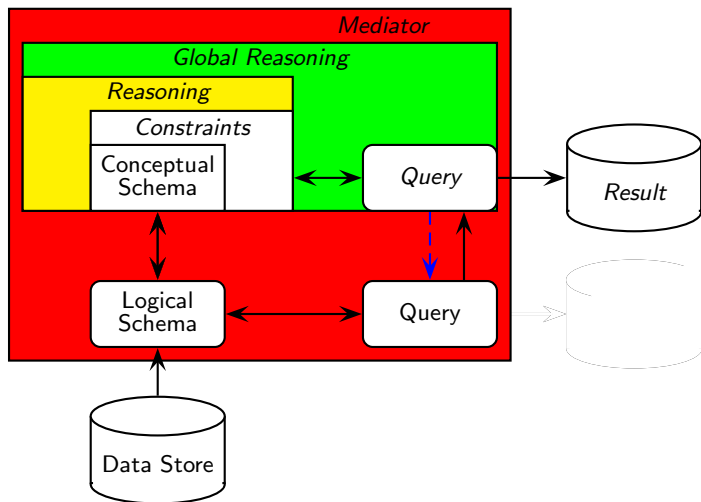
The role of a Conceptual Schema – revisited



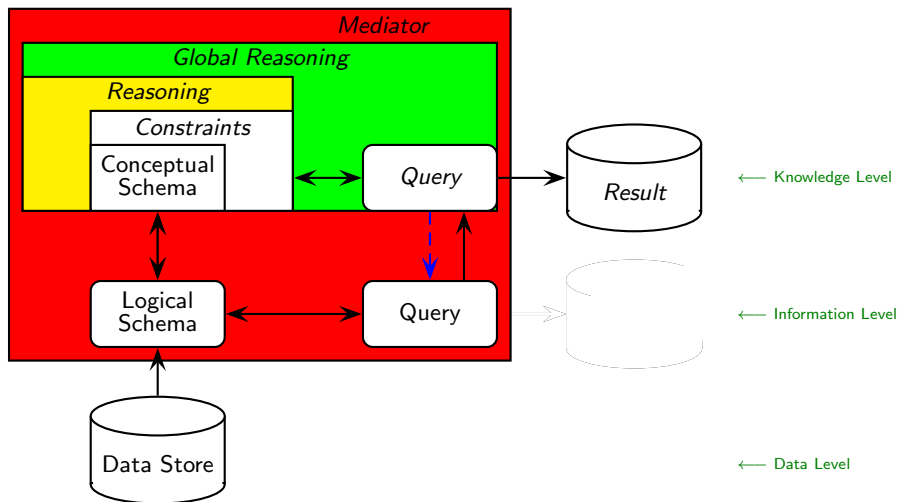
The role of a Conceptual Schema – revisited



The role of a Conceptual Schema – revisited



The role of a Conceptual Schema – revisited



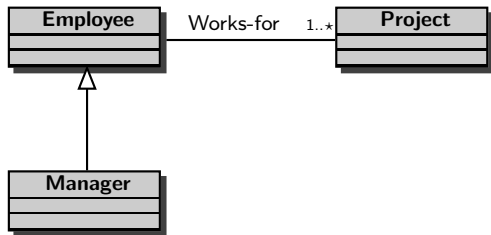
Queries via Conceptual Schemas: the DB assumption

- ▶ Basic assumption: **consistent** information with respect to the constraints introduced by the conceptual schema
- ▶ DB assumption: **complete information** about **each term** appearing in the conceptual schema
- ▶ *Problem*: answer a query over the conceptual schema vocabulary

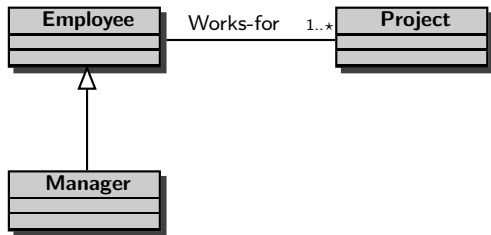
Queries via Conceptual Schemas: the DB assumption

- ▶ Basic assumption: **consistent** information with respect to the constraints introduced by the conceptual schema
- ▶ DB assumption: **complete information about each term** appearing in the conceptual schema
- ▶ *Problem*: answer a query over the conceptual schema vocabulary
- ▶ *Solution*: use a standard DB technology (e.g., SQL, datalog, etc)

Example with DB assumption



Example with DB assumption



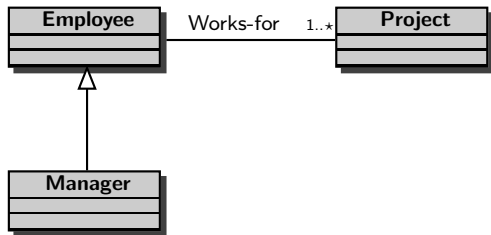
Employee = { John, Mary, Paul }

Manager = { John, Paul }

Works-for = { ⟨John,Prj-A⟩, ⟨Mary,Prj-B⟩ }

Project = { Prj-A, Prj-B }

Example with DB assumption



Employee = { John, Mary, Paul }

Manager = { John, Paul }

Works-for = { ⟨John, Prj-A⟩, ⟨Mary, Prj-B⟩ }

Project = { Prj-A, Prj-B }

$Q(X) :- \text{Manager}(X), \text{Works-for}(X,Y), \text{Project}(Y)$

$\implies \{ \text{John} \}$

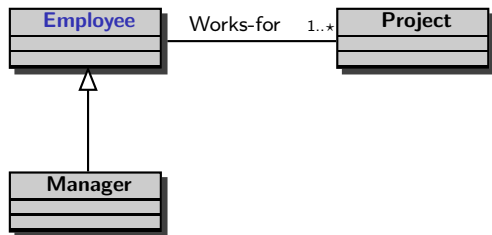
Weakening the DB assumption

- ▶ The DB assumption is against the principle that a conceptual schema presents a richer vocabulary than the data stores (i.e., it plays the role of an ontology).

Weakening the DB assumption

- ▶ The DB assumption is against the principle that a conceptual schema presents a richer vocabulary than the data stores (i.e., it plays the role of an ontology).
- ▶ Partial DB assumption (or conceptual schema with *DBox*): **complete information** about some term appearing in the conceptual schema
- ▶ Standard DB technologies do not apply
- ▶ The query answering problem in this context is inherently complex

Example with partial DB assumption

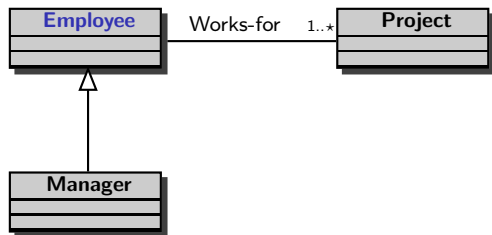


Manager = { John, Paul }

Works-for = { ⟨John, Prj-A⟩, ⟨Mary, Prj-B⟩ }

Project = { Prj-A, Prj-B }

Example with partial DB assumption



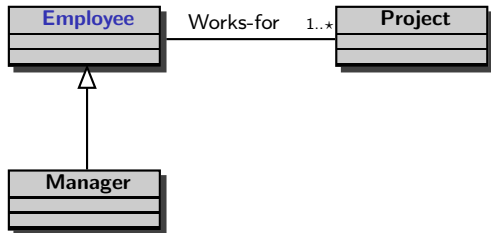
Manager = { John, Paul }

Works-for = { ⟨John, Prj-A⟩, ⟨Mary, Prj-B⟩ }

Project = { Prj-A, Prj-B }

$Q(X) :- \text{Employee}(X)$

Example with partial DB assumption



Manager = { John, Paul }

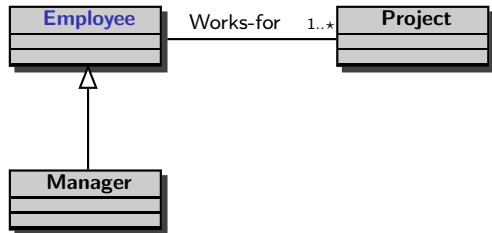
Works-for = { ⟨John, Prj-A⟩, ⟨Mary, Prj-B⟩ }

Project = { Prj-A, Prj-B }

$Q(X) :- \text{Employee}(X)$

$\implies \{ \text{John, Paul, Mary} \}$

Example with partial DB assumption



Manager = { John, Paul }

Works-for = { ⟨John, Prj-A⟩, ⟨Mary, Prj-B⟩ }

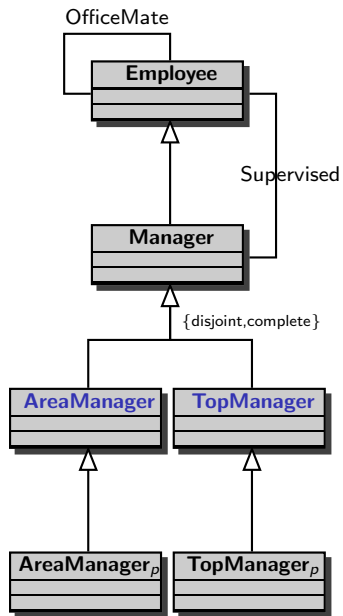
Project = { Prj-A, Prj-B }

$Q(X) :- \text{Employee}(X)$

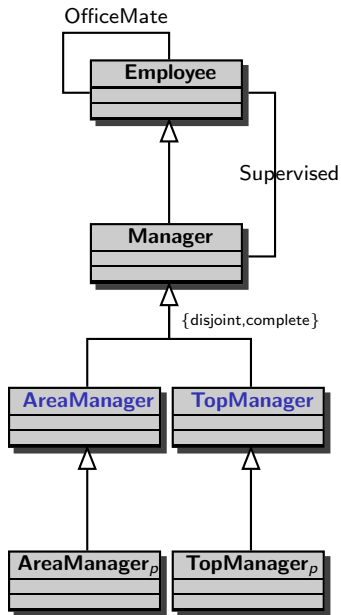
$\Rightarrow \{ \text{John, Paul, Mary} \}$

$\Rightarrow Q'(X) :- \text{Manager}(X) \cup \text{Works-for}(X,Y)$

Andrea's Example

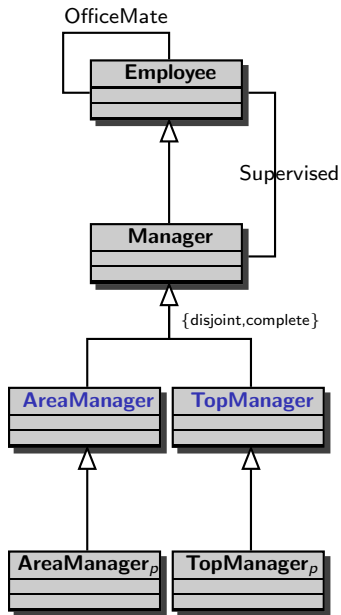


Andrea's Example

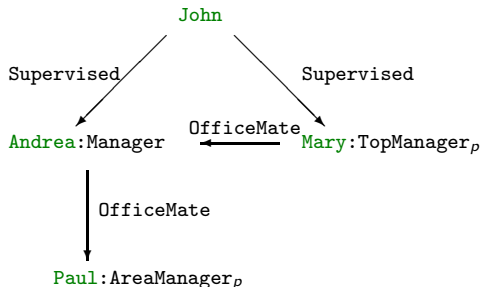


```
Employee = { Andrea, Paul, Mary, John }  
Manager = { Andrea, Paul, Mary }  
AreaManagerp = { Paul }  
TopManagerp = { Mary }  
Supervised = { ⟨John,Andrea⟩, ⟨John,Mary⟩ }  
OfficeMate = { ⟨Mary,Andrea⟩, ⟨Andrea,Paul⟩ }
```

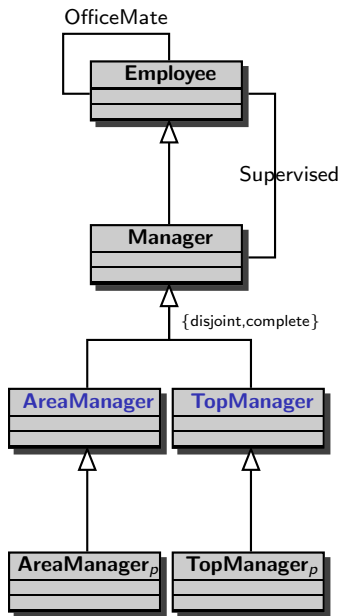

Andrea's Example



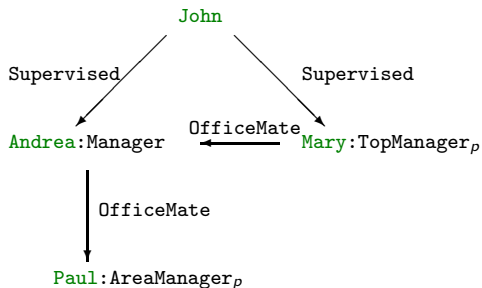
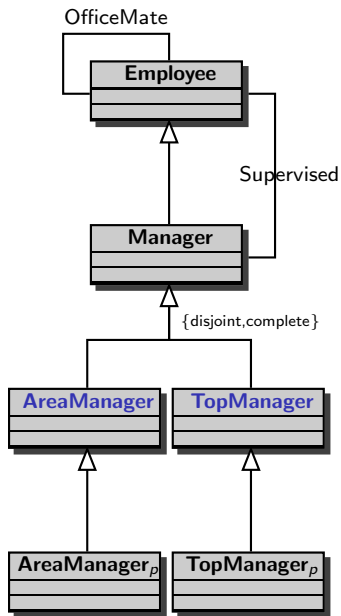
$Employee = \{ \text{Andrea, Paul, Mary, John} \}$
 $Manager = \{ \text{Andrea, Paul, Mary} \}$
 $AreaManager_p = \{ \text{Paul} \}$
 $TopManager_p = \{ \text{Mary} \}$
 $Supervised = \{ \langle \text{John, Andrea} \rangle, \langle \text{John, Mary} \rangle \}$
 $OfficeMate = \{ \langle \text{Mary, Andrea} \rangle, \langle \text{Andrea, Paul} \rangle \}$



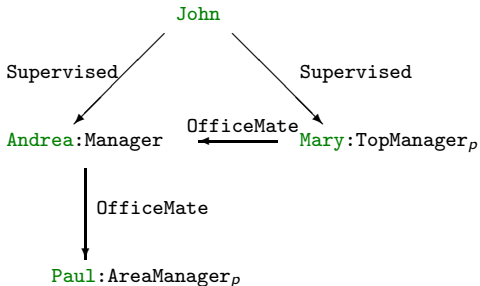
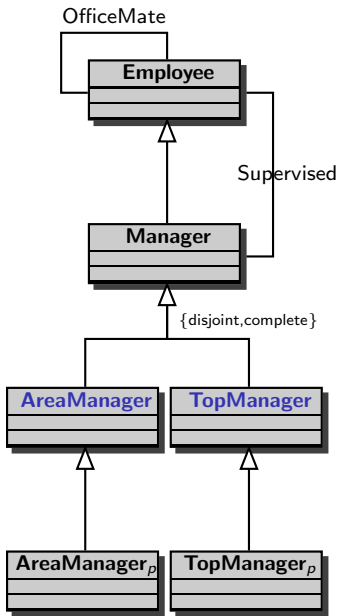
Andrea's Example (cont.)



Andrea's Example (cont.)

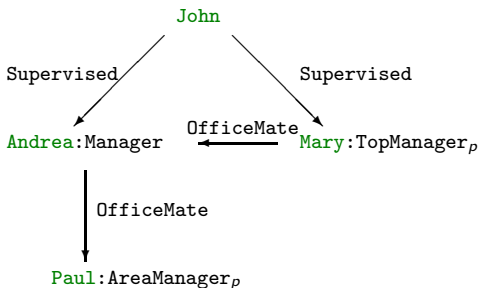
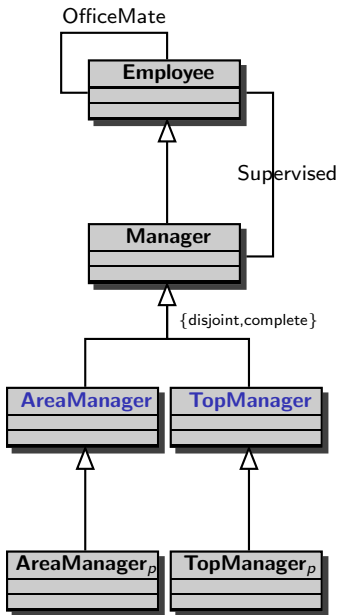


Andrea's Example (cont.)



$Q(X) \text{ :- Supervised}(X,Y), \text{ TopManager}(Y),$
 $\text{ Officemate}(Y,Z), \text{ AreaManager}(Z)$

Andrea's Example (cont.)



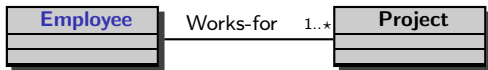
$Q(X) \text{ :- Supervised}(X,Y), \text{ TopManager}(Y),$
 $\text{ Officemate}(Y,Z), \text{ AreaManager}(Z)$

$\Rightarrow \{ \text{John} \}$

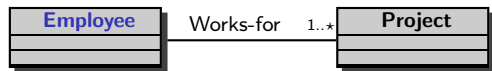
Partial incomplete DB assumption

- ▶ Partial DB assumption (or conceptual schema with *DBox*): **complete information** about *some term* appearing in the conceptual schema
- ▶ Partial incomplete DB assumption (or conceptual schema with *ABox*): **incomplete information** about *some term* appearing in the conceptual schema
- ▶ The partial incomplete DB assumption (conceptual schema with *ABox*) is crucial in data integration scenarios.

Partial incomplete DB assumption



Partial incomplete DB assumption

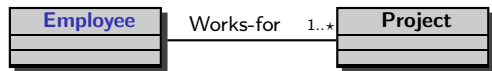


DBox:

Works-for = { $\langle \text{John}, \text{Prj-A} \rangle$, $\langle \text{Mary}, \text{Prj-A} \rangle$ }

Project = { Prj-A, Prj-B }

Partial incomplete DB assumption



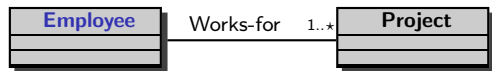
DBox:

Works-for = { $\langle \text{John}, \text{Prj-A} \rangle, \langle \text{Mary}, \text{Prj-A} \rangle$ }

Project = { Prj-A, Prj-B }

\Rightarrow **INCONSISTENT**

Partial incomplete DB assumption



DBox:

Works-for = { $\langle \text{John}, \text{Prj-A} \rangle$, $\langle \text{Mary}, \text{Prj-A} \rangle$ }

Project = { Prj-A, Prj-B }

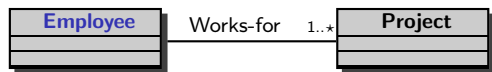
\Rightarrow **INCONSISTENT**

ABox:

Works-for \supseteq { $\langle \text{John}, \text{Prj-A} \rangle$, $\langle \text{Mary}, \text{Prj-A} \rangle$ }

Project \supseteq { Prj-A, Prj-B }

Partial incomplete DB assumption

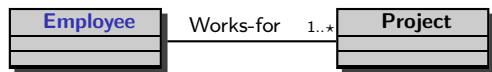


ABox:

Works-for \supseteq { \langle John,Prj-A \rangle , \langle Mary,Prj-A \rangle }

Project \supseteq { Prj-A, Prj-B }

Partial incomplete DB assumption



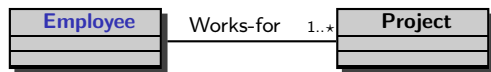
ABox:

Works-for \supseteq { \langle John,Prj-A \rangle , \langle Mary,Prj-A \rangle }

Project \supseteq { Prj-A, Prj-B }

$Q(X)$:- Works-for(Y,X)

Partial incomplete DB assumption



ABox:

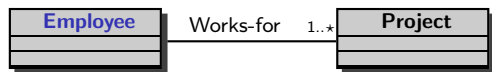
$\text{Works-for} \supseteq \{ \langle \text{John}, \text{Prj-A} \rangle, \langle \text{Mary}, \text{Prj-A} \rangle \}$

$\text{Project} \supseteq \{ \text{Prj-A}, \text{Prj-B} \}$

$Q(X) \text{ :- Works-for}(Y, X)$

$\implies \{ \text{Prj-A}, \text{Prj-B} \}$

Partial incomplete DB assumption



ABox:

Works-for \supseteq { \langle John,Prj-A \rangle , \langle Mary,Prj-A \rangle }

Project \supseteq { Prj-A, Prj-B }

$Q(X) :-$ Works-for(Y,X)

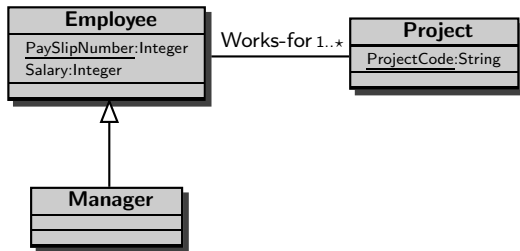
\implies { Prj-A, Prj-B }

\implies $Q'(X) :-$ Project(X) \cup Works-for(Y,X)

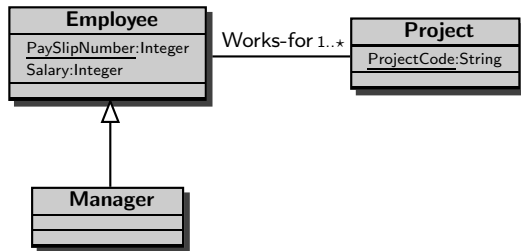
View based Query Processing

- ▶ Mappings between the conceptual schema terms and the information source terms are not necessarily atomic.
- ▶ **Mappings** can be given in terms of a set of **ABox** (or **DBox**) **facts**:
 - ▶ **GAV** (*global-as-view*): ABox (or DBox) facts over the information source vocabulary are associated to terms in the conceptual schema
 - ▶ both the DB and the partial DB assumptions are special cases of GAV
 - ▶ an ER schema can be easily mapped to its corresponding relational schema in some normal form via a GAV mapping
 - ▶ **LAV** (*local-as-view*): a ABox or DBox over the conceptual schema vocabulary is associated to each term in the information source;
 - ▶ **GLAV**: mix of the above.
- ▶ It is non-trivial, even in the pure GAV setting - which is wrongly believed to be computable by simple view unfolding.

Sound GAV mapping



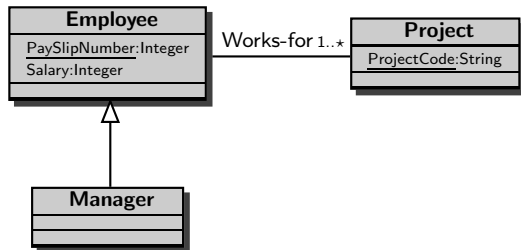
Sound GAV mapping



1-Employee(PaySlipNumber, Salary, ManagerP)

2-Works-for(PaySlipNumber, ProjectCode)

Sound GAV mapping



1-Employee(PaySlipNumber, Salary, ManagerP)

2-Works-for(PaySlipNumber, ProjectCode)

Employee(X) :- 1-Employee(X, Y, false)

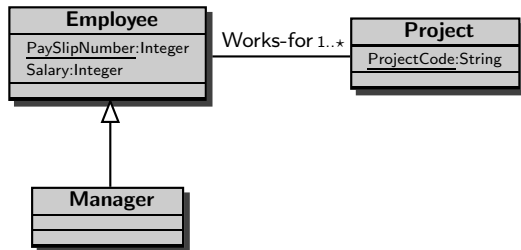
Manager(X) :- 1-Employee(X, Y, true)

Project(Y) :- 2-Works-for(X, Y)

Works-for(X, Y) :- 2-Works-for(X, Y)

Salary(X, Y) :- 1-Employee(X, Y, Z)

Sound GAV mapping



1-Employee(PaySlipNumber, Salary, ManagerP)

2-Works-for(PaySlipNumber, ProjectCode)

Employee(X) :- 1-Employee(X, Y, false)

Manager(X) :- 1-Employee(X, Y, true)

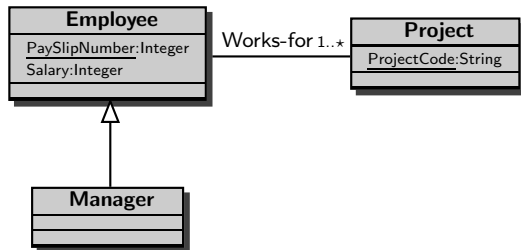
Project(Y) :- 2-Works-for(X, Y)

Works-for(X, Y) :- 2-Works-for(X, Y)

Salary(X, Y) :- 1-Employee(X, Y, Z)

Q(X) :- Employee(X)

Sound GAV mapping

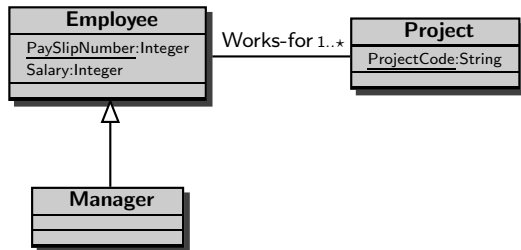


1-Employee(PaySlipNumber, Salary, ManagerP)
2-Works-for(PaySlipNumber, ProjectCode)

Employee(X) :- 1-Employee(X, Y, false) Works-for(X, Y) :- 2-Works-for(X, Y)
Manager(X) :- 1-Employee(X, Y, true) Salary(X, Y) :- 1-Employee(X, Y, Z)
Project(Y) :- 2-Works-for(X, Y)

Q(X) :- Employee(X)
⇒ Q'(X) :- 1-Employee(X, Y, Z) ∪ 2-Works-for(X, W)

Sound GAV mapping



1-Employee(PaySlipNumber, Salary, ManagerP)
2-Works-for(PaySlipNumber, ProjectCode)

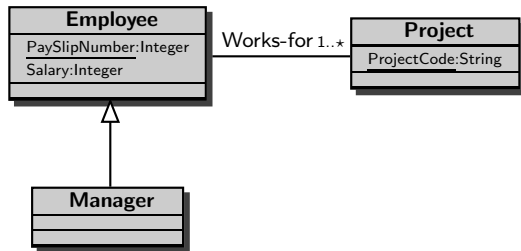
Employee(X) :- 1-Employee(X,Y,false) Works-for(X,Y) :- 2-Works-for(X,Y)
Manager(X) :- 1-Employee(X,Y,true) Salary(X,Y) :- 1-Employee(X,Y,Z)
Project(Y) :- 2-Works-for(X,Y)

Q(X) :- Employee(X)

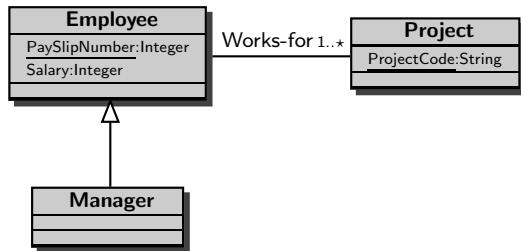
⇒ Q'(X) :- 1-Employee(X,Y,Z) ∪ 2-Works-for(X,W)

← not coming from unfolding!

Sound LAV mapping

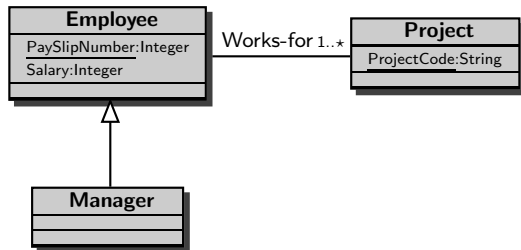


Sound LAV mapping



- 1-Employee(PaySlipNumber, Salary, ManagerP)
- 2-Works-for(PaySlipNumber, ProjectCode)

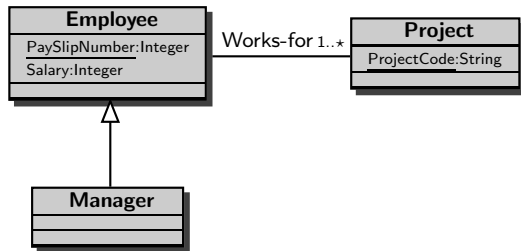
Sound LAV mapping



1-Employee(PaySlipNumber, Salary, ManagerP)
2-Works-for(PaySlipNumber, ProjectCode)

```
1-Employee(X,Y,Z) :- Manager(X), Salary(X,Y), Z=true
1-Employee(X,Y,Z) :- Employee(X), ¬Manager(X), Salary(X,Y), Z=false
2-Works-for(X,Y) :- Works-for(X,Y)
```


Sound LAV mapping



1-Employee(PaySlipNumber, Salary, ManagerP)

2-Works-for(PaySlipNumber, ProjectCode)

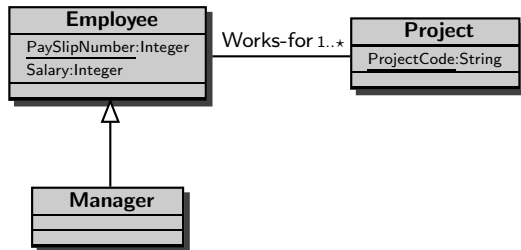
1-Employee(X,Y,Z) :- Manager(X), Salary(X,Y), Z=true

1-Employee(X,Y,Z) :- Employee(X), ¬Manager(X), Salary(X,Y), Z=false

2-Works-for(X,Y) :- Works-for(X,Y)

Q(X) :- Manager(X), Works-for(X,Y), Project(Y)

Sound LAV mapping



1-Employee(PaySlipNumber, Salary, ManagerP)

2-Works-for(PaySlipNumber, ProjectCode)

1-Employee(X,Y,Z) :- Manager(X), Salary(X,Y), Z=true

1-Employee(X,Y,Z) :- Employee(X), ¬Manager(X), Salary(X,Y), Z=false

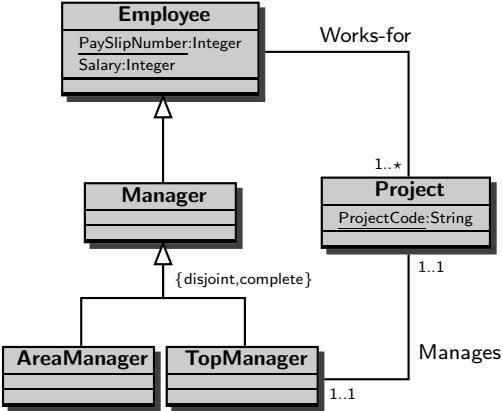
2-Works-for(X,Y) :- Works-for(X,Y)

Q(X) :- Manager(X), Works-for(X,Y), Project(Y)

⇒ Q'(X) :- 1-Employee(X,Y,true), 2-Works-for(X,Z)

Reasoning over queries

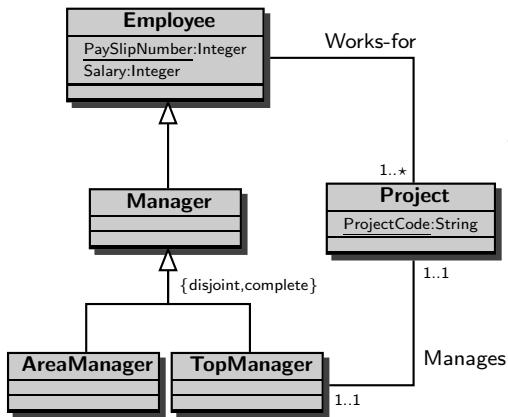
$Q(X,Y) :- \text{Employee}(X), \text{Works-for}(X,Y), \text{Manages}(X,Y)$



$\forall x. \text{Manager}(x) \rightarrow \forall y. \neg \text{WORKS-FOR}(x,y)$

Reasoning over queries

$Q(X, Y) :- \text{Employee}(X), \text{Works-for}(X, Y), \text{Manages}(X, Y)$



$\forall x. \text{Manager}(x) \rightarrow \forall y. \neg \text{WORKS-FOR}(x, y)$



INCONSISTENT QUERY!

Conclusions

Conclusions

Do you want to exploit conceptual schema knowledge
(i.e., an ontology)
in your data intensive application?

Conclusions

Do you want to exploit conceptual schema knowledge
(i.e., an ontology)
in your data intensive application?

Pay attention!