Ontologies and Databases: myths and challenges

Enrico Franconi

Free University of Bozen-Bolzano, Italy

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

Summary

- What is an Ontology
- (Description) Logics for Conceptual Modelling
- Querying a DB via a Conceptual Schema

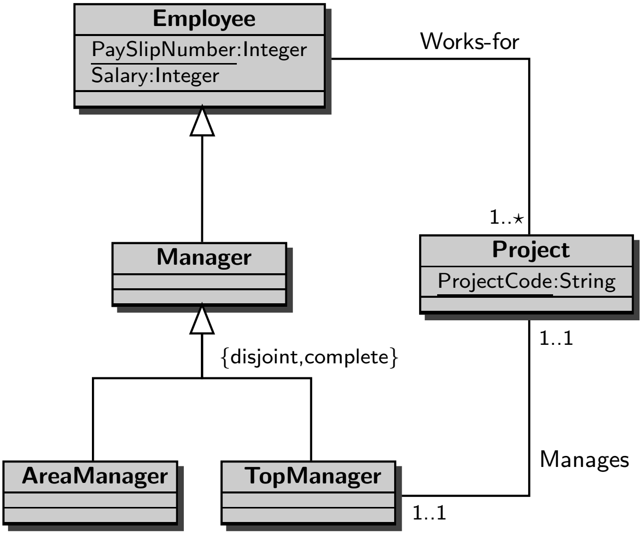
What is an Ontology

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

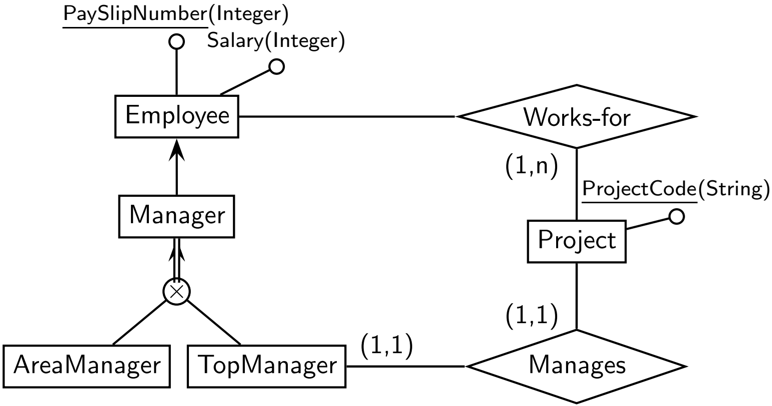
Ontologies and Conceptual Data Models

- 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.

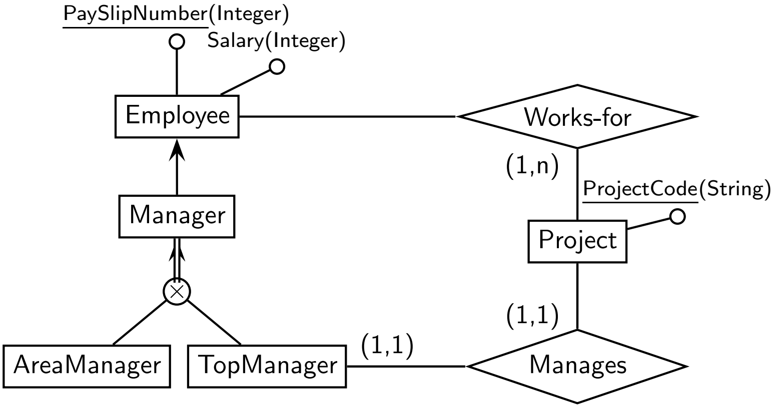
UML Class Diagram



Entity-Relationship Schema

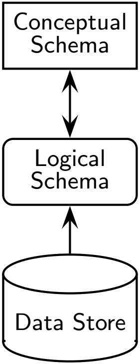


Entity-Relationship Schema

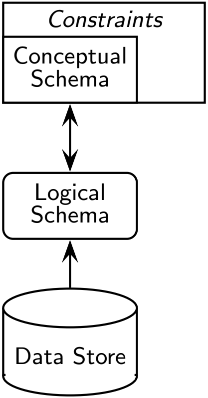


Go to part on *Query Answering*

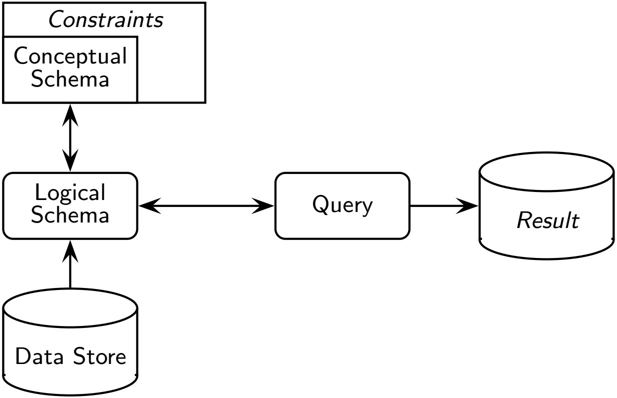
The role of a Conceptual Schema



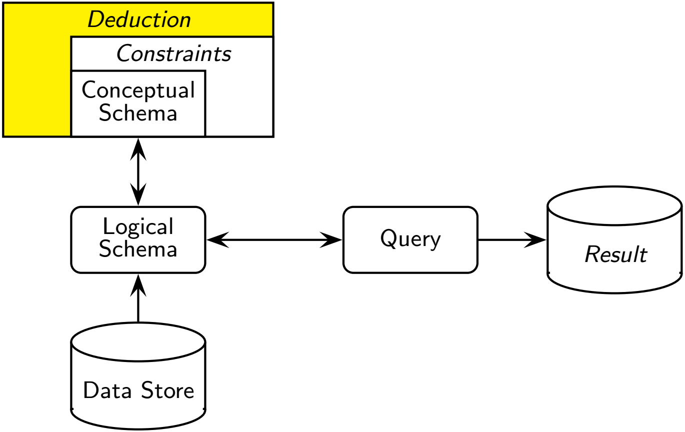
The role of a Conceptual Schema



The role of a Conceptual Schema



The role of a Conceptual Schema

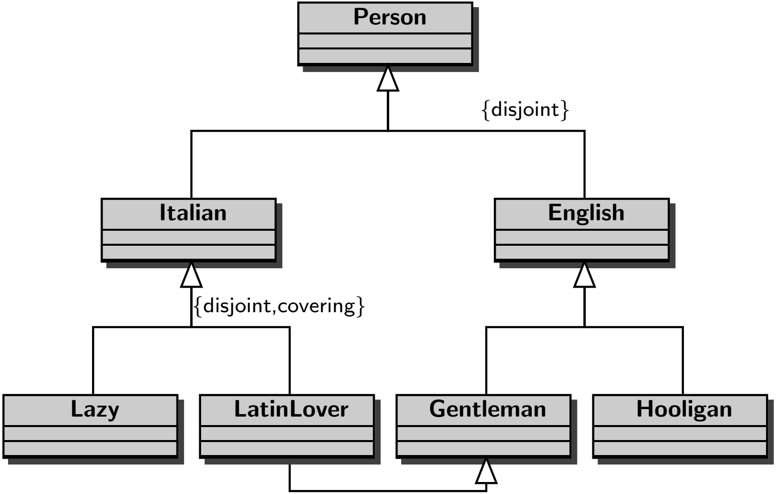


Reasoning

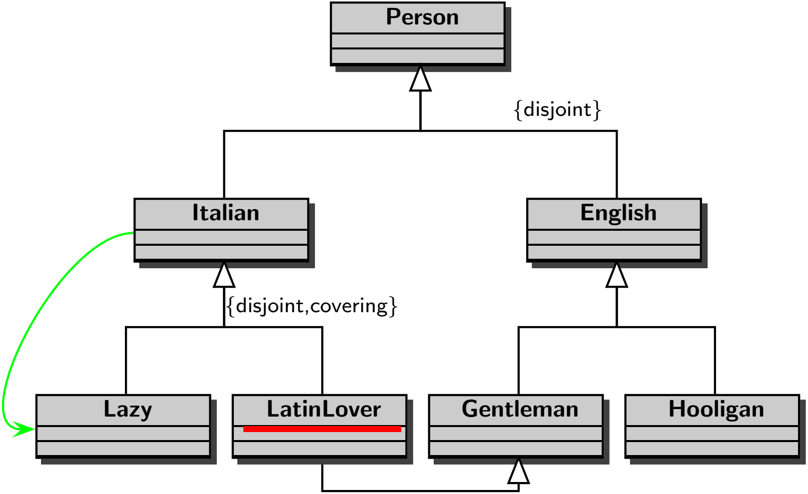
Given an ontology – seen as a collection of constraints – it is possible that additional constraints can be inferred.

- 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

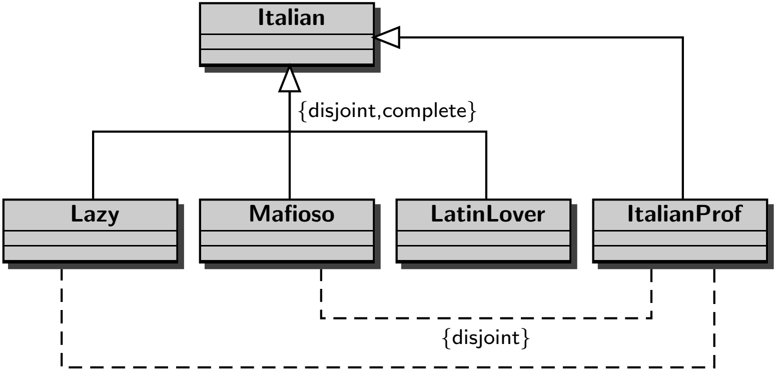


LatinLover $= \emptyset$

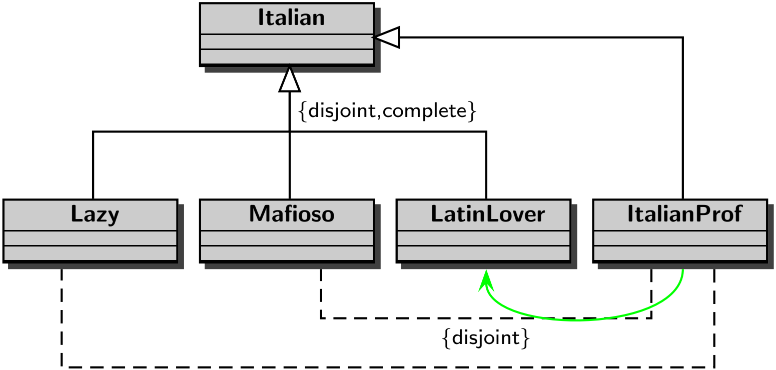
Italian $\subseteq$ Lazy

Italian $\equiv$ Lazy

Reasoning: cute professors



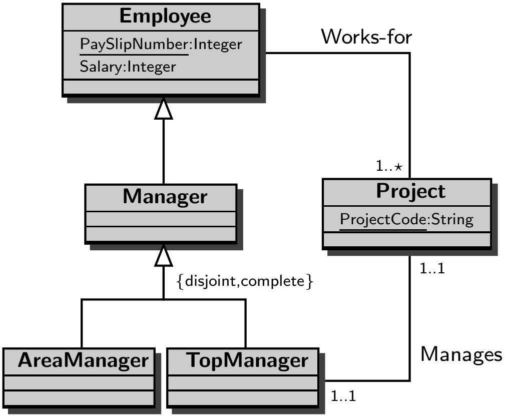
Reasoning: cute professors



implies

ItalianProf $\subseteq$ LatinLover

Reasoning with Conceptual Schemas



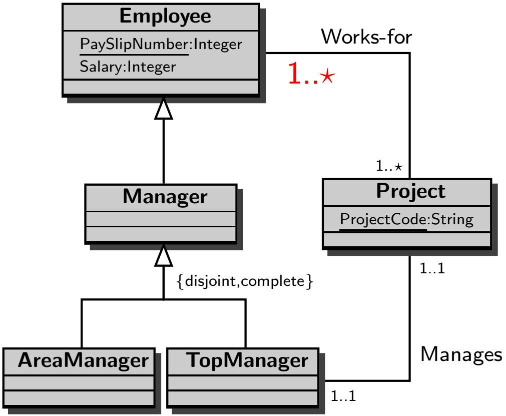
- Managers do not work for a project (she/he just manages it):

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

  $\texttt{Manager} \sqsubseteq \neg\exists \texttt{WORKS-FOR}.\, \top$

  $\texttt{Manager} \subseteq \texttt{Employee} \setminus \pi_1 \texttt{WORKS-FOR}$

Reasoning with Conceptual Schemas



- Managers do not work for a project (she/he just manages it):

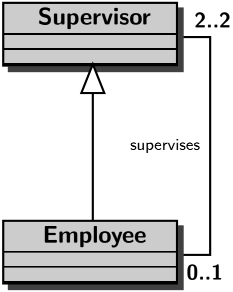
  $\forall x.\, \texttt{Manager}(x) \rightarrow \neg\exists y.\, \texttt{WORKS-FOR}(x, y)$

  $\texttt{Manager} \sqsubseteq \neg\exists \texttt{WORKS-FOR}.\, \top$

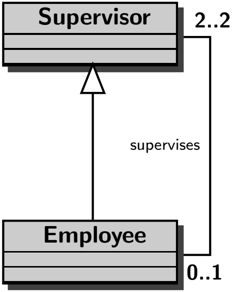
  $\texttt{Manager} \subseteq \texttt{Employee} \setminus \pi_1 \texttt{WORKS-FOR}$

- If the minimum cardinality for the participation of employees to the *works-for* relationship is increased, then ...

The democratic company



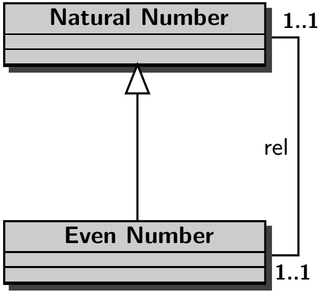
The democratic company



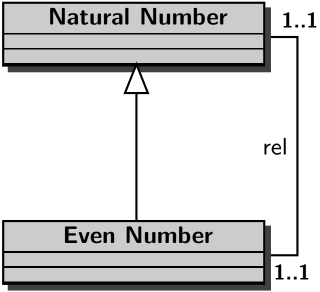
implies
"the classes Employee and Supervisor necessarily contain an infinite number of instances".

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

How many numbers?

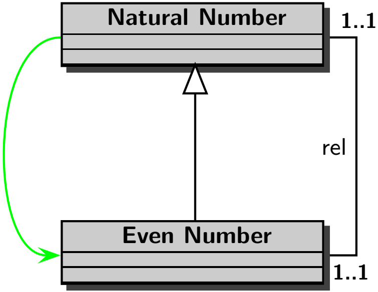


How many numbers?



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

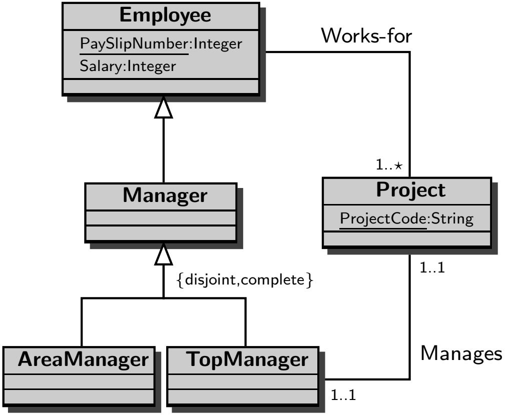
- What is an Ontology
- (Description) Logics for Conceptual Modelling
- Querying a DB via a Conceptual Schema

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., OWL)

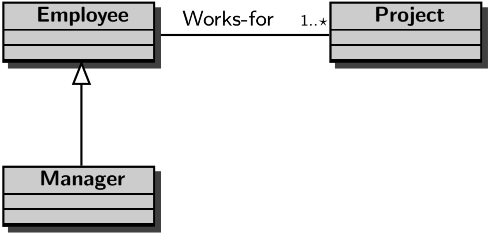
Encoding Conceptual Schemas in (Description) Logics

- 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., 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.



$$
\begin{array}{lll}
\texttt{Works-for} & \sqsubseteq & \texttt{emp}/2 : \texttt{Employee} \sqcap \texttt{act}/2 : \texttt{Project} \\
\texttt{Manages} & \sqsubseteq & \texttt{man}/2 : \texttt{TopManager} \sqcap \texttt{prj}/2 : \texttt{Project} \\
\texttt{Employee} & \sqsubseteq & \exists^{=1}[\texttt{worker}](\texttt{PaySlipNumber} \sqcap \texttt{num}/2 : \texttt{Integer}) \sqcap \\
 & & \exists^{=1}[\texttt{payee}](\texttt{Salary} \sqcap \texttt{amount}/2 : \texttt{Integer}) \\
\top & \sqsubseteq & \exists^{\leq 1}[\texttt{num}](\texttt{PaySlipNumber} \sqcap \texttt{worker}/2 : \texttt{Employee}) \\
\texttt{Manager} & \sqsubseteq & \texttt{Employee} \sqcap (\texttt{AreaManager} \sqcup \texttt{TopManager}) \\
\texttt{AreaManager} & \sqsubseteq & \texttt{Manager} \sqcap \neg \texttt{TopManager} \\
\texttt{TopManager} & \sqsubseteq & \texttt{Manager} \sqcap \exists^{=1}[\texttt{man}]\texttt{Manages} \\
\texttt{Project} & \sqsubseteq & \exists^{\geq 1}[\texttt{act}]\texttt{Works-for} \sqcap \exists^{=1}[\texttt{prj}]\texttt{Manages} \\
\ldots & &
\end{array}
$$

Relational algebra constraints



Employee/1, Manager/1, Project/1, Works-for/2

Manager $\subseteq$ Employee

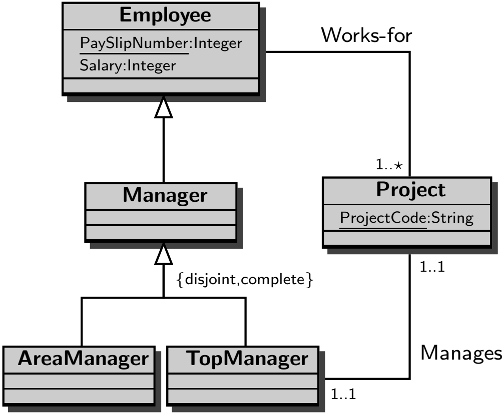
π_1 Works-for $\subseteq$ Employee
π_2 Works-for $\subseteq$ Project

Project $\subseteq$ π_2 Works-for

Set-based Constraints

$\text{Works-for} \subseteq \text{Employee} \times \text{Project}$
$\text{Manages} \subseteq \text{TopManager} \times \text{Project}$
$\text{Employee} \subseteq \{e \mid \sharp(\text{PaySlipNumber} \cap (\{e\} \times \texttt{Integer})) \geq 1\}$
$\text{Employee} \subseteq \{e \mid \sharp(\text{Salary} \cap (\{e\} \times \texttt{Integer})) \geq 1\}$
$\text{Project} \subseteq \{p \mid \sharp(\text{ProjectCode} \cap (\{p\} \times \texttt{String})) \geq 1\}$
$\text{TopManager} \subseteq \{m \mid 1 \geq \sharp(\text{Manages} \cap (\{m\} \times \Omega)) \geq 1\}$
$\text{Project} \subseteq \{p \mid 1 \geq \sharp(\text{Manages} \cap (\Omega \times \{p\})) \geq 1\}$
$\text{Project} \subseteq \{p \mid \sharp(\text{Works-for} \cap (\Omega \times \{p\})) \geq 1\}$
$\text{Manager} \subseteq \text{Employee}$
$\text{AreaManager} \subseteq \text{Manager}$
$\text{TopManager} \subseteq \text{Manager}$
$\text{AreaManager} \cap \text{TopManager} = \emptyset$
$\text{Manager} \subseteq \text{AreaManager} \cup \text{TopManager}$

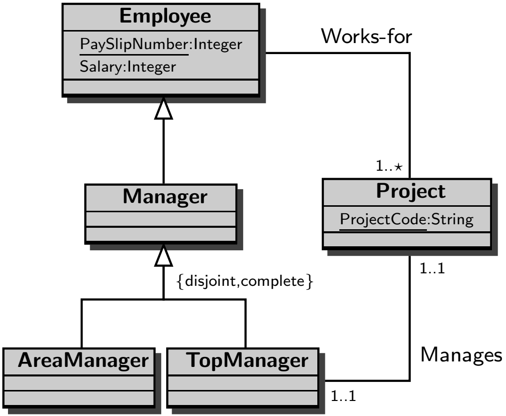
Deducing constraints



Managers are employees who do not work for a project (she/he just manages it):

$\texttt{Employee} \sqcap \neg(\exists^{\geq 1}[\texttt{emp}]\texttt{Works-for}) \sqsubseteq \texttt{Manager}, \quad \texttt{Manager} \sqsubseteq \neg(\exists^{\geq 1}[\texttt{emp}]\texttt{Works-for})$

Deducing constraints



Managers are employees who do not work for a project (she/he just manages it):

$\texttt{Employee} \sqcap \neg(\exists^{\geq 1}[\texttt{emp}]\texttt{Works-for}) \sqsubseteq \texttt{Manager}, \quad \texttt{Manager} \sqsubseteq \neg(\exists^{\geq 1}[\texttt{emp}]\texttt{Works-for})$

$\models$ For every project, there is at least one employee who is not a manager:

$\texttt{Project} \sqsubseteq \exists^{\geq 1}[\texttt{act}](\texttt{Works-for} \sqcap \texttt{emp} : \neg\texttt{Manager})$

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 $\mathcal{DLR}$ inference engine;
- i•com verifies the specification, infers implicit facts and stricter constraints, and manifests any inconsistencies during the conceptual modelling phase.

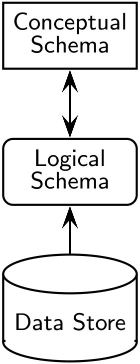
Next on "Myths and Challenges":

- What is an Ontology
- (Description) Logics for Conceptual Modelling
- Querying a DB via a Conceptual Schema
  - We will see how an ontology can play the role of a "mediator" wrapping a (source) database.
  - Examples will show how apparently simple cases are not easy.
  - We will learn about view-based query processing with GAV and LAV mappings.
  - We introduce the difference between closed world and open world semantics in this context.
  - We will see how only the closed world semantics should be used while using ontologies to wrap databases, in order for the mediated system to behave like a database (black-box metaphor)
  - We will see that the data complexity of query answering can be beyond the one of SQL.

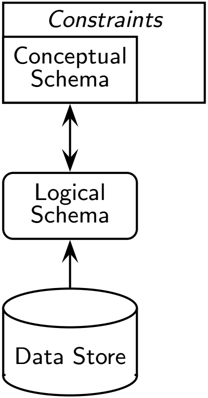
Summary

- What is an Ontology
- (Description) Logics for Conceptual Modelling
- Querying a DB via a Conceptual Schema

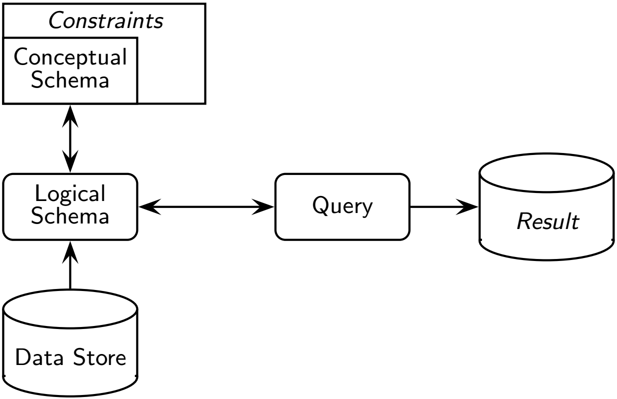
The role of a Conceptual Schema



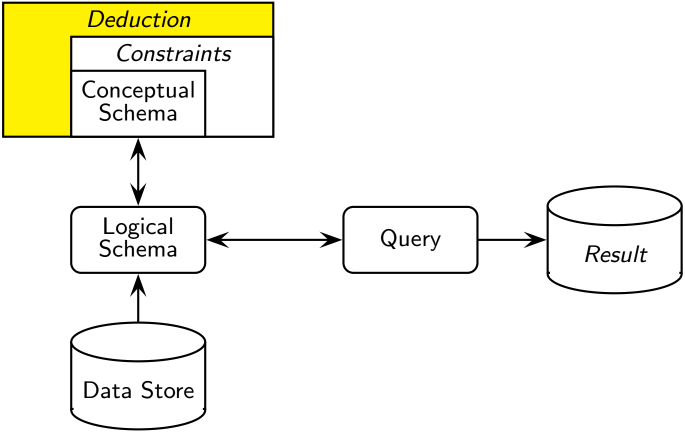
The role of a Conceptual Schema



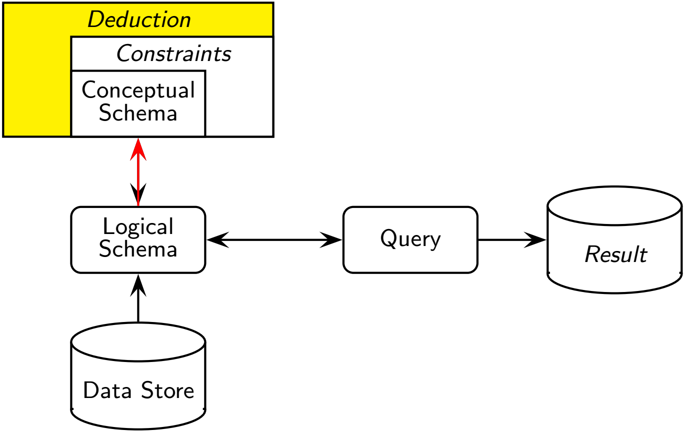
The role of a Conceptual Schema



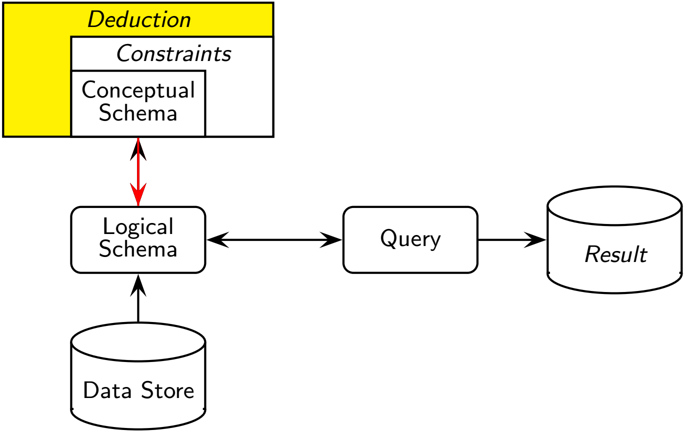
The role of a Conceptual Schema



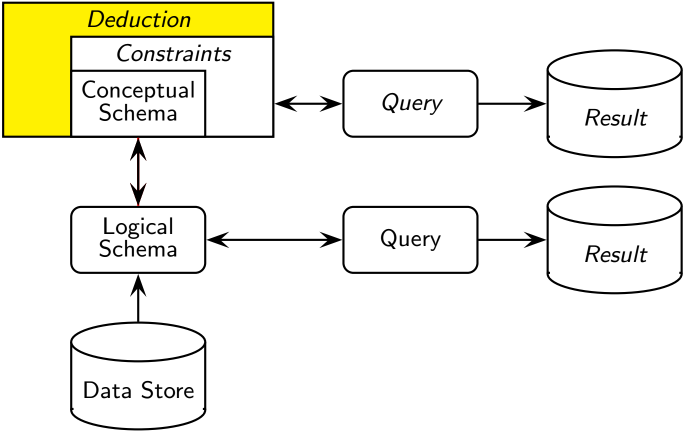
The role of a Conceptual Schema



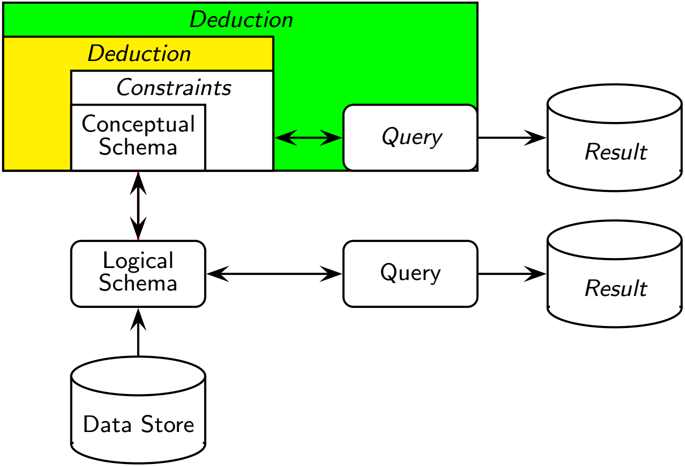
The role of a Conceptual Schema



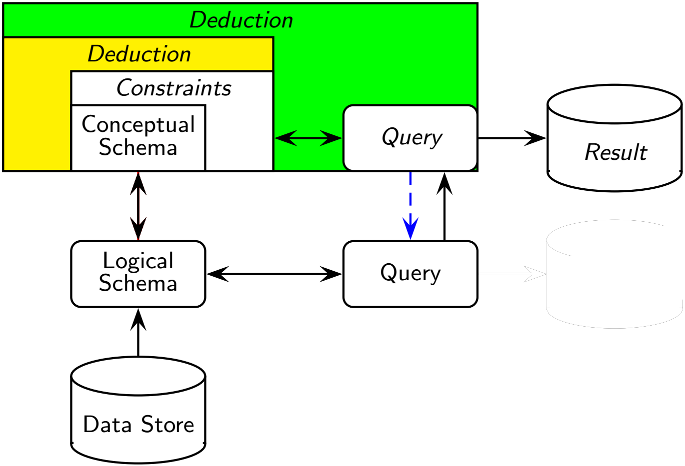
The role of a Conceptual Schema



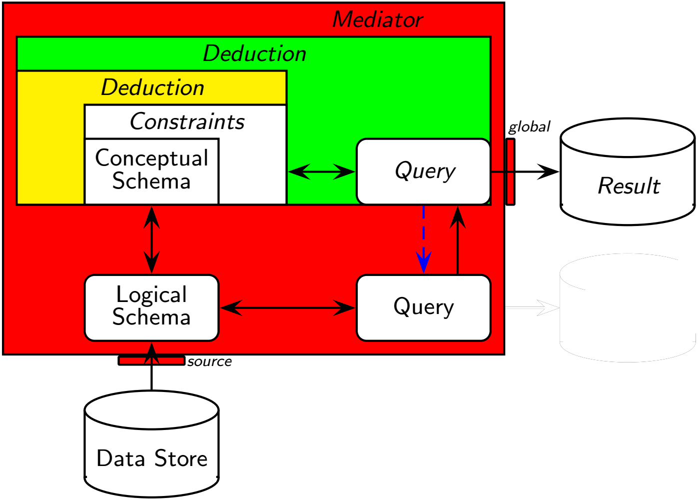
The role of a Conceptual Schema



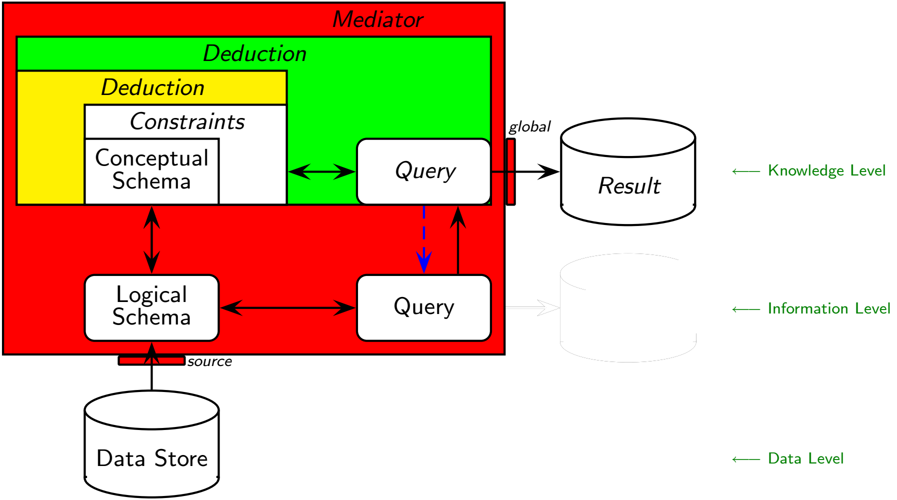
The role of a Conceptual Schema



The role of a Conceptual Schema



The role of a Conceptual Schema



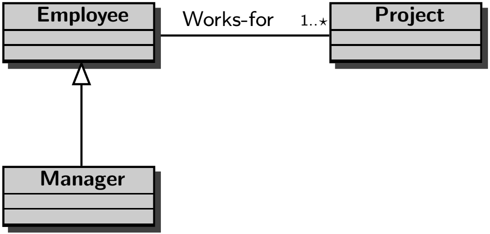
Queries via Conceptual Schemas: the DB assumption

- 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

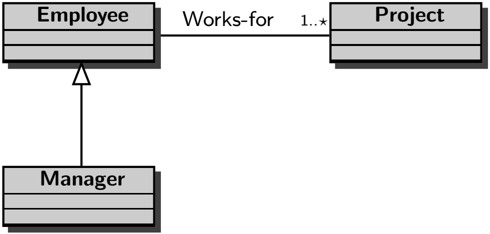
Queries via Conceptual Schemas: the DB assumption

- 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



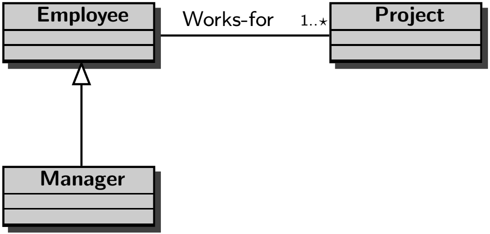
`Employee = {` John, Mary, Paul `}`

`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) :- Manager(X), Works-for(X,Y), Project(Y)
$\Longrightarrow$ { John }

Partial 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

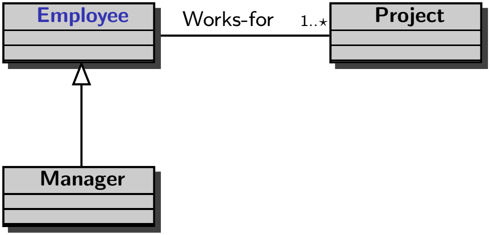
- The DB assumption is against the principle that a conceptual schema presents a richer vocabulary than the data stores (i.e., it plays the role of an ontology).
- Partial DB assumption (or conceptual schema with *exact views*): complete information about *some* term appearing in the conceptual schema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

Partial 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 *exact views*): 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

We are dealing now with an *incomplete database*

Example with partial DB assumption

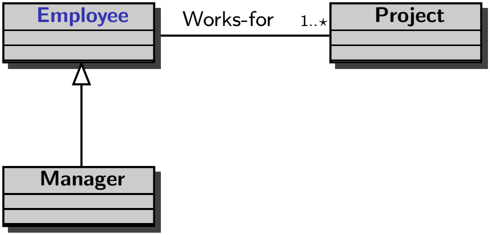


Manager = { John, Paul }

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

Project = { Prj-A, Prj-B }

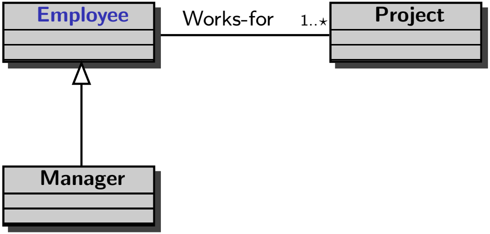
Example with partial DB assumption



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

Q(X) :- Employee(X)

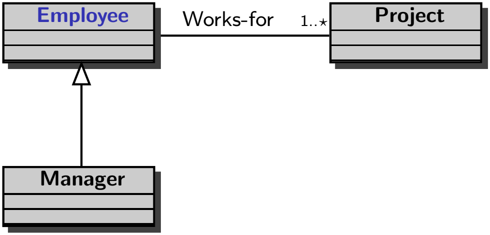
Example with partial DB assumption



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

Q(X) :- Employee(X)
⟹ { John, Paul, Mary }

Example with partial DB assumption

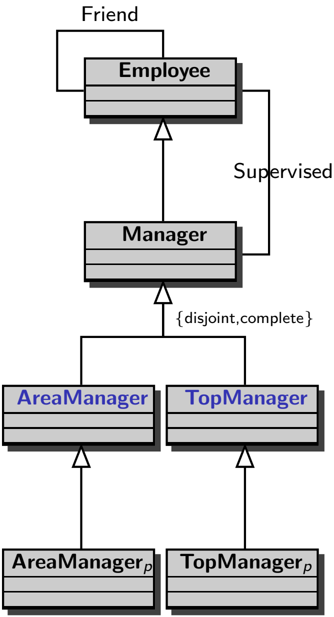


Manager = { John, Paul }
Works-for = { ⟨John,Prj-A⟩, ⟨Mary,Prj-B⟩ }
Project = { Prj-A, Prj-B }

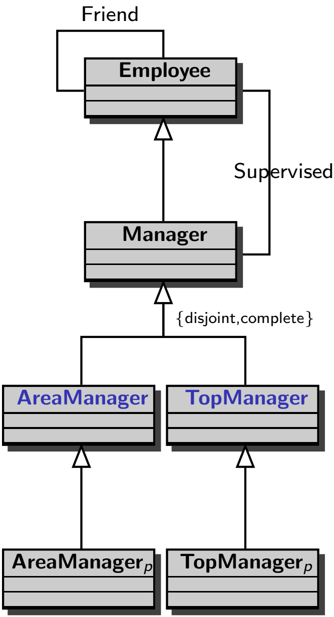
Q(X) :- Employee(X)
⟹ { John, Paul, Mary }

⟹ Q'(X) :- Manager(X) ∪ Works-for(X,Y)

Andrea's Example

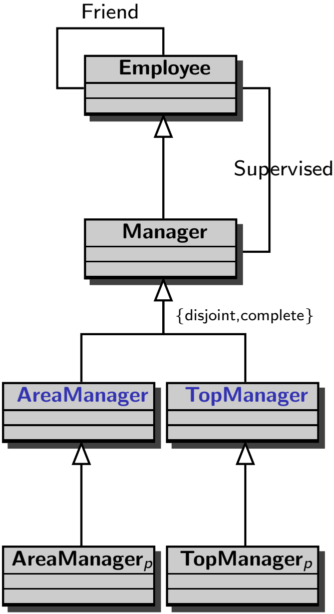


Andrea's Example

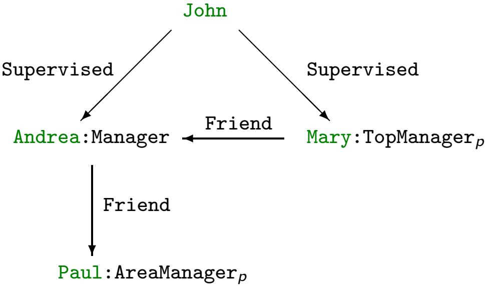



```
Employee = { Andrea, Paul, Mary, John }
Manager = { Andrea, Paul, Mary}
AreaManager_p = { Paul }
TopManager_p = { Mary }
Supervised = { ⟨John,Andrea⟩, ⟨John,Mary⟩ }
Friend = { ⟨Mary,Andrea⟩, ⟨Andrea,Paul⟩ }
```

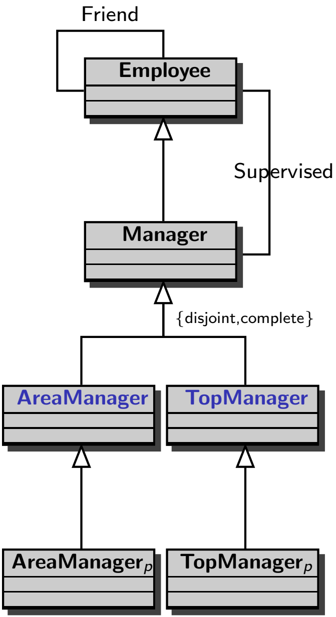
Andrea's Example



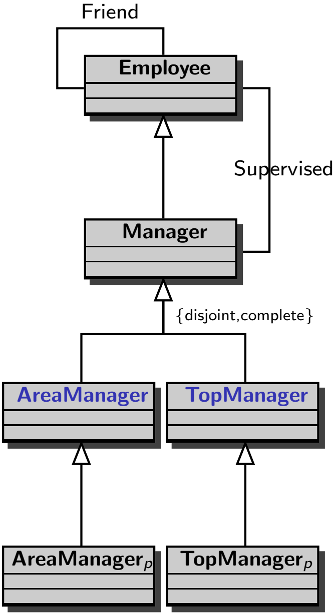
Employee = { Andrea, Paul, Mary, John }
Manager = { Andrea, Paul, Mary}
AreaManager$_p$ = { Paul }
TopManager$_p$ = { Mary }
Supervised = { ⟨John,Andrea⟩, ⟨John,Mary⟩ }
Friend = { ⟨Mary,Andrea⟩, ⟨Andrea,Paul⟩ }

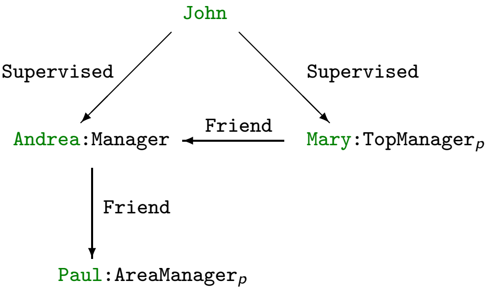


Andrea's Example (cont.)

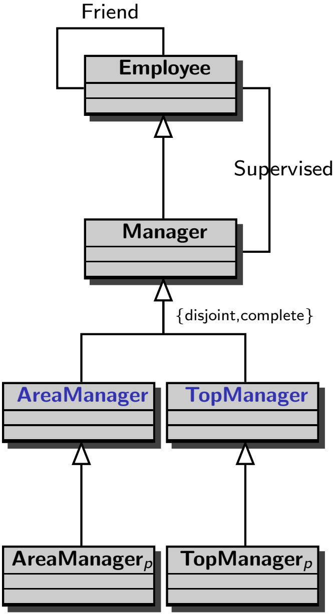


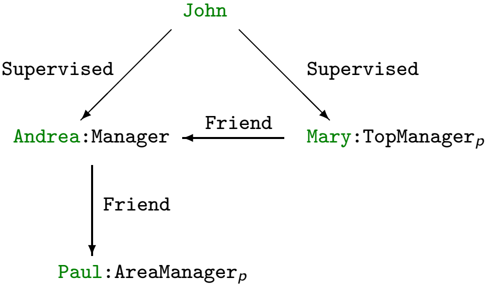
Andrea's Example (cont.)





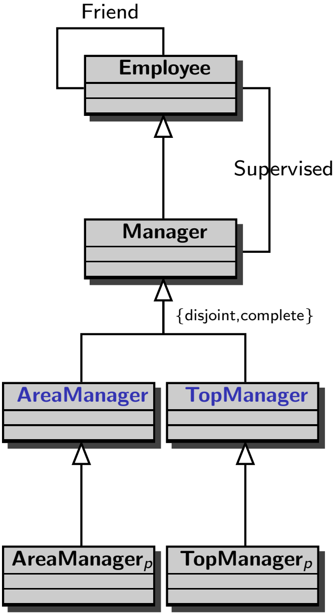
Andrea's Example (cont.)

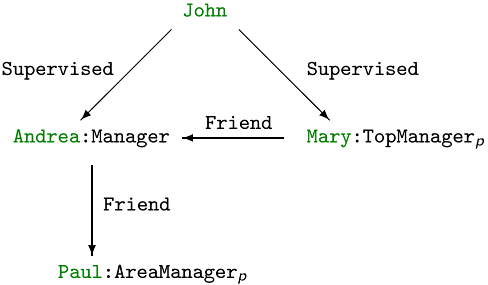





```
Q :- Supervised(John,Y), TopManager(Y),
     Friend(Y,Z), AreaManager(Z)
```

Andrea's Example (cont.)





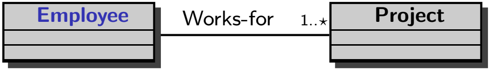
Q :- Supervised(John,Y), TopManager(Y),
Friend(Y,Z), AreaManager(Z)

$\Longrightarrow$ YES

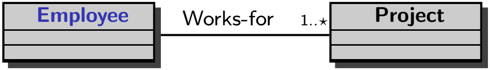
Partial incomplete DB assumption

1. DB assumption (aka constraints over a database): complete information about *all* terms appearing in the conceptual schema
2. Partial DB assumption (aka conceptual schema with ***exact views***): complete information about *some* term appearing in the conceptual schema
3. Partial incomplete DB assumption (aka conceptual schema with ***sound views***): incomplete information about *some* term appearing in the conceptual schema; this is also called an ABox
   - The partial incomplete DB assumption (conceptual schema with sound views) is said to be crucial in data integration scenarios.

Partial incomplete DB assumption



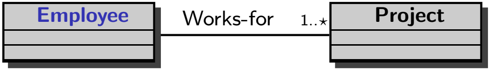
Partial incomplete DB assumption



Partial DB assumption (exact views):

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

Partial incomplete DB assumption

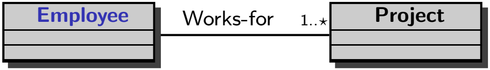


Partial DB assumption (exact views):

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

$\Longrightarrow$ INCONSISTENT

Partial incomplete DB assumption



Partial DB assumption (exact views):

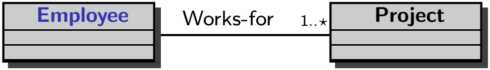
Works-for = { ⟨John,Prj-A⟩, ⟨Mary,Prj-A⟩ }
Project = { Prj-A, Prj-B }

$\Longrightarrow$ INCONSISTENT

Partial incomplete DB assumption (sound views):

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

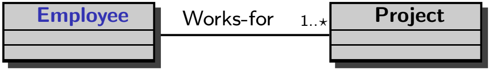
Querying with sound views



Partial incomplete DB assumption (sound views), i.e., an *ABox*:

$\texttt{Works-for} \supseteq \{ \langle \texttt{John}, \texttt{Prj-A} \rangle, \langle \texttt{Mary}, \texttt{Prj-A} \rangle \}$
$\texttt{Project} \supseteq \{ \texttt{Prj-A}, \texttt{Prj-B} \}$

Querying with sound views

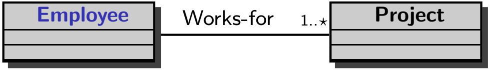


Partial incomplete DB assumption (sound views), i.e., an *ABox*:

Works-for $\supseteq$ { ⟨John,Prj-A⟩, ⟨Mary,Prj-A⟩ }
Project $\supseteq$ { Prj-A, Prj-B }

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

Querying with sound views

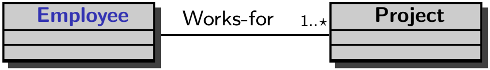


Partial incomplete DB assumption (sound views), i.e., an *ABox*:

`Works-for` $\supseteq$ { ⟨`John`,`Prj-A`⟩, ⟨`Mary`,`Prj-A`⟩ }
`Project` $\supseteq$ { `Prj-A`, `Prj-B` }

`Q(X) :- Works-for(Y,X)`
$\Longrightarrow$ { `Prj-A`, `Prj-B` }

Querying with sound views



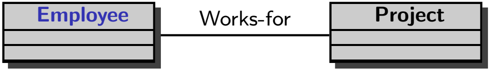
Partial incomplete DB assumption (sound views), i.e., an *ABox*:

Works-for $\supseteq$ { ⟨John,Prj-A⟩, ⟨Mary,Prj-A⟩ }
Project $\supseteq$ { Prj-A, Prj-B }

Q(X) :- Works-for(Y,X)
$\Longrightarrow$ { Prj-A, Prj-B }

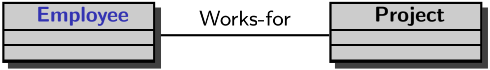
$\Longrightarrow$ Q'(X) :- Project(X) $\cup$ Works-for(Y,X)

Exact vs Sound views



- Additional constraint as a standard view over the data:
  $\texttt{Bad-Project} = \texttt{Project} \setminus \pi_2\texttt{Works-for}$
  $\forall \texttt{x}.\ \texttt{Bad-Project(x)} \leftrightarrow \texttt{Project(x)} \land \neg \exists \texttt{y.Works-for(y,x)}$
  $\texttt{Bad-Project} = \texttt{Project} \sqcap \neg \exists \texttt{Works-for}^{-}.\top$

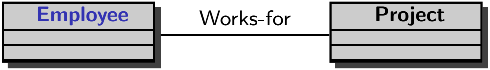
Exact vs Sound views



- Additional constraint as a standard view over the data:
  Bad-Project = Project $\setminus$ π_2Works-for
  $\forall$x. Bad-Project(x)$\leftrightarrow$ Project(x)$\wedge\neg\exists$y.Works-for(y,x)
  Bad-Project = Project$\sqcap\neg\exists$Works-for$^{-}.\top$
- exact views:
  Works-for = { ⟨John,Prj-A⟩, ⟨Mary,Prj-A⟩ }
  Project = { Prj-A, Prj-B }
- Q(X) :- Bad-Project(X)

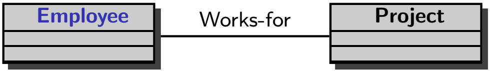
- sound views:
  Works-for $\supseteq$ { ⟨John,Prj-A⟩, ⟨Mary,Prj-A⟩ }
  Project $\supseteq$ { Prj-A, Prj-B }
- Q(X) :- Bad-Project(X)

Exact vs Sound views



- Additional constraint as a standard view over the data:
  $\texttt{Bad-Project} = \texttt{Project} \setminus \pi_2\texttt{Works-for}$
  $\forall \texttt{x}.\ \texttt{Bad-Project(x)} \leftrightarrow \texttt{Project(x)} \land \neg\exists \texttt{y}.\texttt{Works-for(y,x)}$
  $\texttt{Bad-Project} = \texttt{Project} \sqcap \neg\exists \texttt{Works-for}^{-}.\top$
- exact views:
  $\texttt{Works-for} = \{ \langle \texttt{John,Prj-A} \rangle, \langle \texttt{Mary,Prj-A} \rangle \}$
  $\texttt{Project} = \{ \texttt{Prj-A, Prj-B} \}$
- `Q(X) :- Bad-Project(X)`
  $\Longrightarrow \{ \texttt{Prj-B} \}$
- sound views:
  $\texttt{Works-for} \supseteq \{ \langle \texttt{John,Prj-A} \rangle, \langle \texttt{Mary,Prj-A} \rangle \}$
  $\texttt{Project} \supseteq \{ \texttt{Prj-A, Prj-B} \}$
- `Q(X) :- Bad-Project(X)`

Exact vs Sound views



- Additional constraint as a standard view over the data:

  $\texttt{Bad-Project} = \texttt{Project} \setminus \pi_2 \texttt{Works-for}$

  $\forall x.\ \texttt{Bad-Project}(x) \leftrightarrow \texttt{Project}(x) \wedge \neg \exists y.\texttt{Works-for}(y,x)$

  $\texttt{Bad-Project} = \texttt{Project} \sqcap \neg \exists \texttt{Works-for}^{-}.\top$

- exact views:

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

  $\texttt{Project} = \{\ \texttt{Prj-A},\ \texttt{Prj-B}\ \}$

- Q(X) :- Bad-Project(X)

  $\Longrightarrow \{\ \texttt{Prj-B}\ \}$

- sound views:

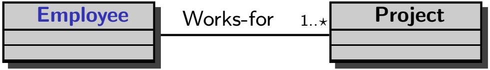
  $\texttt{Works-for} \supseteq \{\ \langle \texttt{John}, \texttt{Prj-A} \rangle,\ \langle \texttt{Mary}, \texttt{Prj-A} \rangle\ \}$

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

- Q(X) :- Bad-Project(X)

  $\Longrightarrow \{\ \}$ *does not scale down to standard DB answer!*

Compositionality of Queries

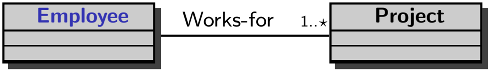


- sound views:

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

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

Compositionality of Queries



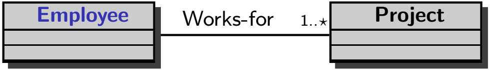
- sound views:

  Works-for $\supseteq$ { ⟨John,Prj-A⟩ }
  Project $\supseteq$ { Prj-A, Prj-B }

- Query as a standard view over the data:

  Q(X) :- Works-for(Y,X) $\quad Q = \pi_2$Works-for

Compositionality of Queries



- sound views:

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

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

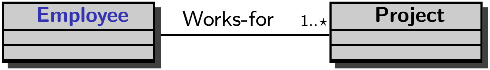
- Query as a standard view over the data:

  $\texttt{Q(X) :- Works-for(Y,X)} \qquad \texttt{Q} = \pi_2 \texttt{Works-for}$

  - $\texttt{Q} = \texttt{EVAL}(\pi_2 \texttt{Works-for})$

  - $\texttt{Q} = \pi_2(\texttt{EVAL}(\texttt{Works-for}))$

Compositionality of Queries



- sound views:

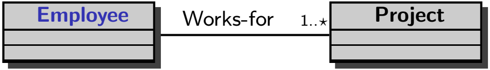
  $\texttt{Works-for} \supseteq \{ \langle \texttt{John}, \texttt{Prj-A} \rangle \}$
  $\texttt{Project} \supseteq \{ \texttt{Prj-A}, \texttt{Prj-B} \}$

- Query as a standard view over the data:

  $\texttt{Q(X) :- Works-for(Y,X)} \qquad \texttt{Q} = \pi_2 \texttt{Works-for}$

  - $\texttt{Q} = \texttt{EVAL}(\pi_2 \texttt{Works-for})$
    $\Longrightarrow \{ \texttt{Prj-A}, \texttt{Prj-B} \}$
  - $\texttt{Q} = \pi_2(\texttt{EVAL}(\texttt{Works-for}))$

Compositionality of Queries



- sound views:

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

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

- Query as a standard view over the data:

  $\texttt{Q(X) :- Works-for(Y,X)} \qquad \texttt{Q} = \pi_2 \texttt{Works-for}$

  - $\texttt{Q} = \texttt{EVAL}(\pi_2 \texttt{Works-for})$

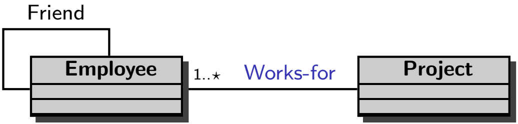
    $\Longrightarrow \{ \texttt{Prj-A}, \texttt{Prj-B} \}$

  - $\texttt{Q} = \pi_2(\texttt{EVAL}(\texttt{Works-for}))$

    $\Longrightarrow \{ \texttt{Prj-A} \}$

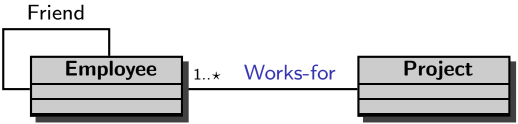
Queries are not compositional wrt certain answer semantics!

Complexity of Query answering



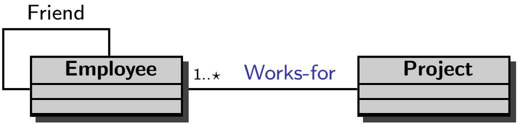
- exact views:
  Friend = {⟨John,Mary⟩,...}; Employee = {John,Mary,...}
  Project = { Prj-A, Prj-B, Prj-C }

Complexity of Query answering



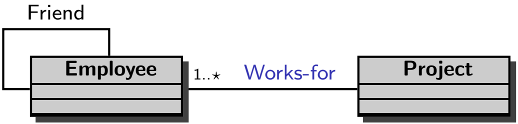
- exact views:
  `Friend` $=$ {⟨`John`,`Mary`⟩,...}; `Employee` $=$ {`John`,`Mary`,...}
  `Project` $=$ { `Prj-A`, `Prj-B`, `Prj-C` }
- `Q :- Works-for(E1,P), Works-for(E2,P), Friend(E1,E2).`
  Is it unavoidable that there are two friends working for the same project?

Complexity of Query answering



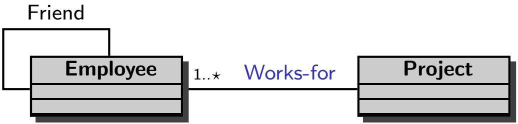
- exact views:
  Friend = {⟨John,Mary⟩,...}; Employee = {John,Mary,...}
  Project = { Prj-A, Prj-B, Prj-C }
- Q :- Works-for(E1,P), Works-for(E2,P), Friend(E1,E2).
  Is it unavoidable that there are two friends working for the same project?
  - YES: in any legal database instance, there are at least two friends working for the same project.

Complexity of Query answering



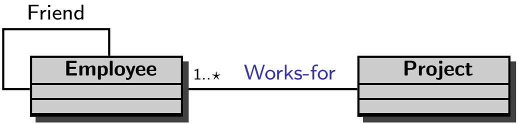
- exact views:
  Friend = {⟨John,Mary⟩,...}; Employee = {John,Mary,...}
  Project = { Prj-A, Prj-B, Prj-C }
- Q :- Works-for(E1,P), Works-for(E2,P), Friend(E1,E2).
  Is it unavoidable that there are two friends working for the same project?
  - YES: in any legal database instance, there are at least two friends working for the same project.
  - NO: there is at least a legal database instance in which no two friends work for the same project.

Complexity of Query answering



- exact views:
  Friend = {⟨John,Mary⟩,...}; Employee = {John,Mary,...}
  Project = { Prj-A, Prj-B, Prj-C }
- Q :- Works-for(E1,P), Works-for(E2,P), Friend(E1,E2).
  Is it unavoidable that there are two friends working for the same project?
  - YES: in any legal database instance, there are at least two friends working for the same project.
  - NO: there is at least a legal database instance in which no two friends work for the same project.
  - With *sound views* the answer is always NO, since there is at least a legal database instance with *enough* distinct projects so that no two friends work for the same project.

Complexity of Query answering



- exact views:
  Friend = {⟨John,Mary⟩,...}; Employee = {John,Mary,...}
  Project = { Prj-A, Prj-B, Prj-C }
- Q :- Works-for(E1,P), Works-for(E2,P), Friend(E1,E2).
  Is it unavoidable that there are two friends working for the same project?
  - YES: in any legal database instance, there are at least two friends working for the same project.
  - NO: there is at least a legal database instance in which no two friends work for the same project.
  - With *sound views* the answer is always NO, since there is at least a legal database instance with *enough* distinct projects so that no two friends work for the same project.

Query answering with exact views is np-hard in data complexity (3-col), and it is strictly harder than with sound views (ABoxes)!

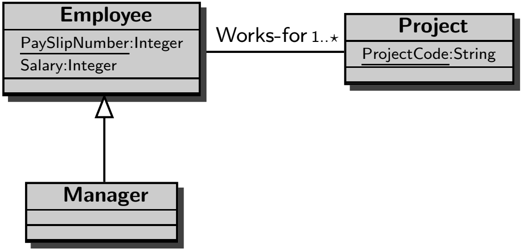
Expressive Ontology Languages

- Exact views as nominals.

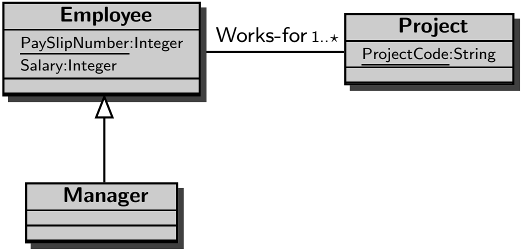
View based Query Processing

- 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 sound (or exact) views:
  - GAV (*global-as-view*): sound (or exact) views 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 sound or exact view 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.
- It is mostly studied with sound views, due to the negative complexity results with exact views discussed before.

Sound GAV mapping

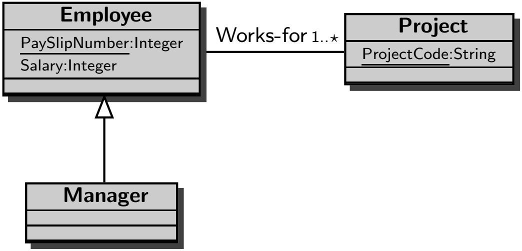


Sound GAV mapping



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

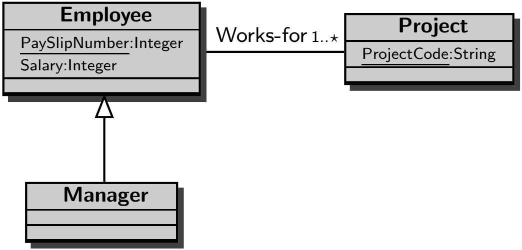
Sound GAV mapping




```
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

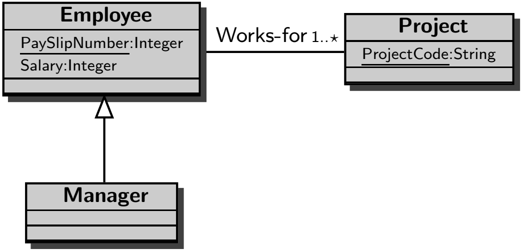



```
1-Employee(PaySlipNumber,Salary,ManagerP)
2-Works-for(PaySlipNumber,ProjectCode)
```

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

Sound GAV mapping

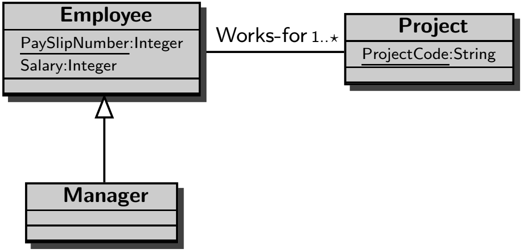



```
1-Employee(PaySlipNumber,Salary,ManagerP)
2-Works-for(PaySlipNumber,ProjectCode)
```

```
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



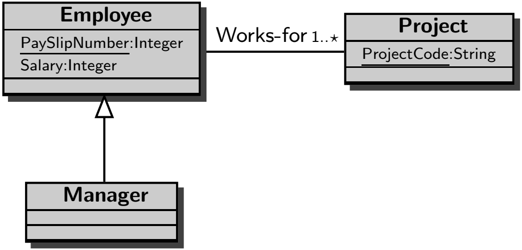

```
1-Employee(PaySlipNumber,Salary,ManagerP)
2-Works-for(PaySlipNumber,ProjectCode)
```

```
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)
```

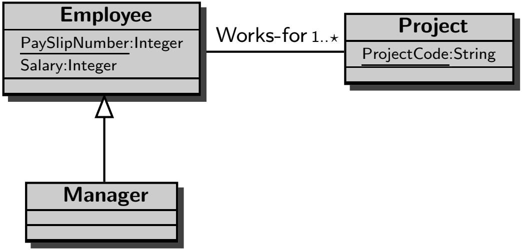
```
Q(X) :- Employee(X)
```

$\Longrightarrow$ `Q'(X) :- 1-Employee(X,Y,Z)` $\cup$ `2-Works-for(X,W)` $\leftarrow$ not coming from unfolding!

Sound LAV mapping

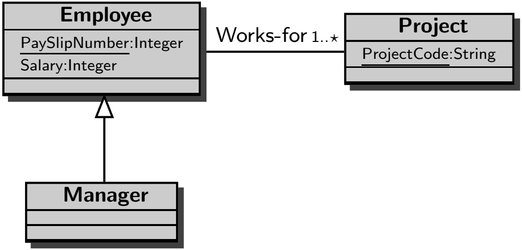


Sound LAV mapping



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

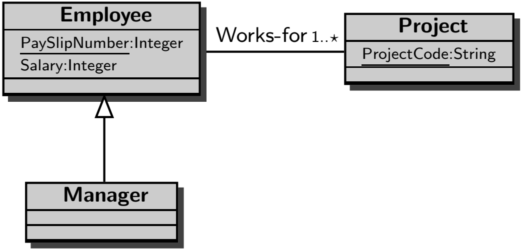
Sound LAV mapping




```
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

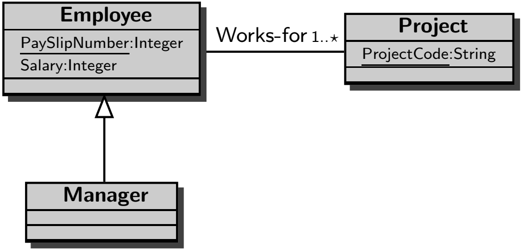



```
1-Employee(PaySlipNumber,Salary,ManagerP)
2-Works-for(PaySlipNumber,ProjectCode)
```

```
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



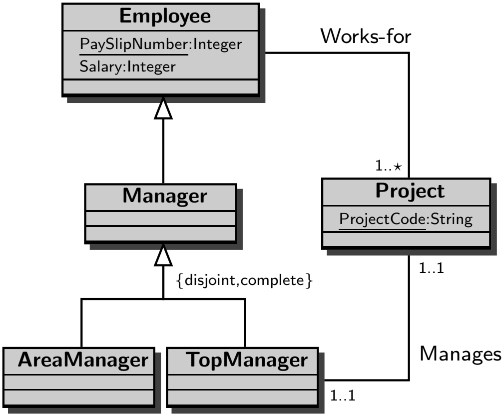

```
1-Employee(PaySlipNumber,Salary,ManagerP)
2-Works-for(PaySlipNumber,ProjectCode)

      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) :- Employee(X), Works-for(X,Y), Manages(X,Y)`



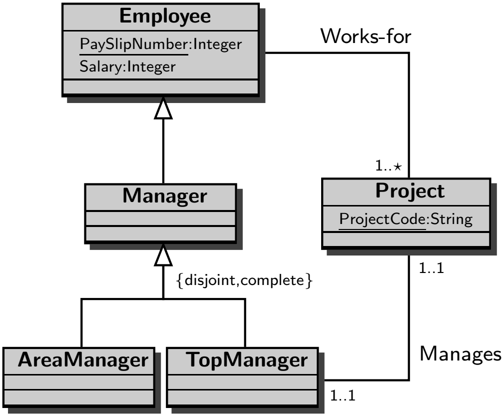
$\forall x.\, \texttt{Manager}(x) \rightarrow \neg\exists y.\, \texttt{WORKS-FOR}(x, y)$

$\texttt{Manager} \sqsubseteq \neg\exists\texttt{WORKS-FOR}.\top$

$\texttt{Manager} \subseteq \texttt{Employee} \setminus \pi_1\texttt{WORKS-FOR}$

Reasoning over queries

`Q(X,Y) :- Employee(X), Works-for(X,Y), Manages(X,Y)`



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

$\texttt{Manager} \sqsubseteq \neg\exists \texttt{WORKS-FOR}.\top$

$\texttt{Manager} \subseteq \texttt{Employee} \setminus \pi_1 \texttt{WORKS-FOR}$

$\leadsto$ INCONSISTENT QUERY!

Conclusions

Conclusions

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

Conclusions

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

Pay attention!