Conceptual schemas and ontologies for database access: myths and challenges

KRDB Research Centre
Knowledge Representation meets DataBases

Enrico Franconi
Free University of Bozen-Bolzano, Italy

Auckland, 7 November 2007

http://www.inf.unibz.it/∼franconi

Summary

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

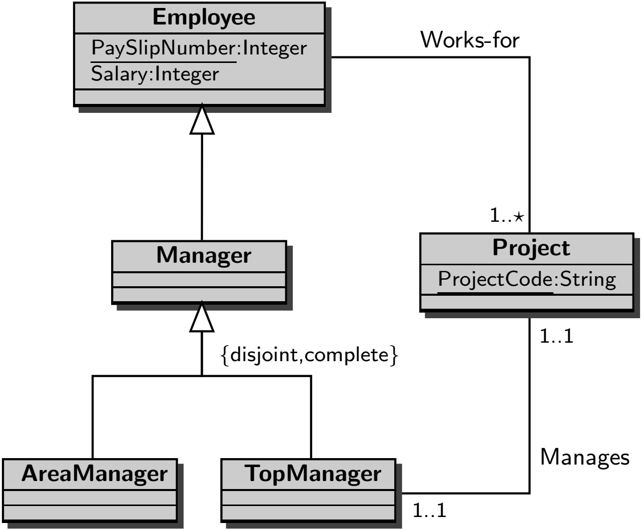
What is an Ontology

- 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.

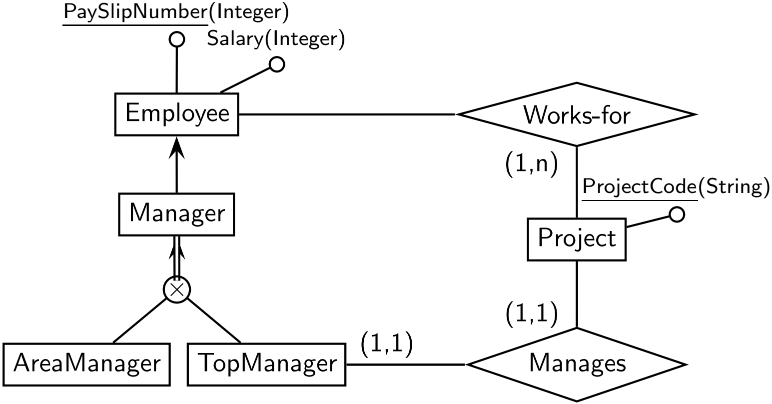
Ontologies and Conceptual Data Models

- 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.

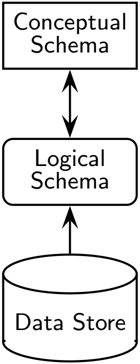
UML Class Diagram



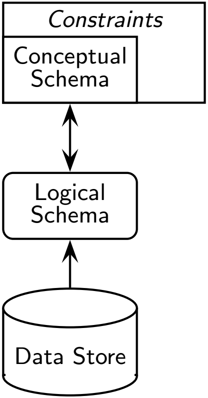
Entity-Relationship Schema



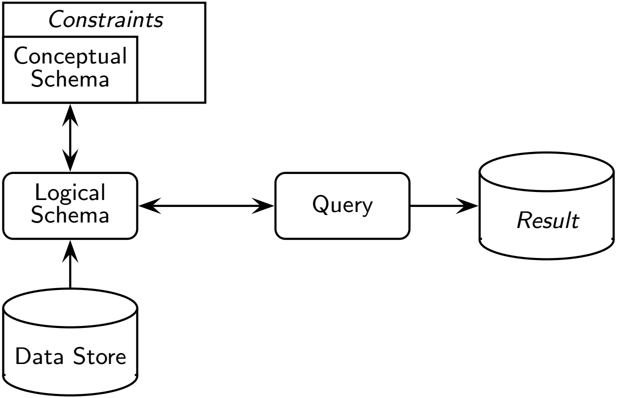
The role of a Conceptual Schema



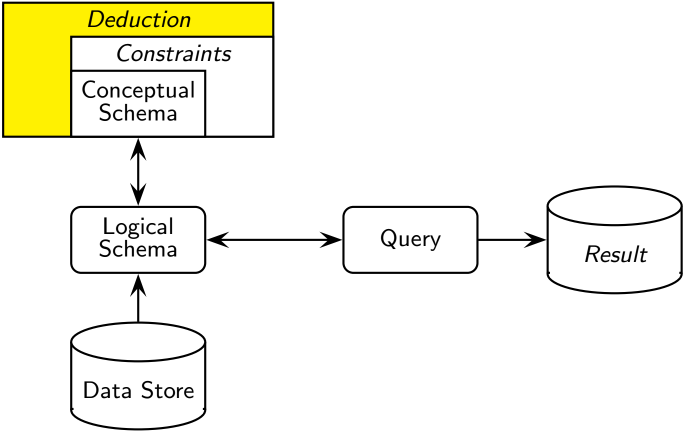
The role of a Conceptual Schema



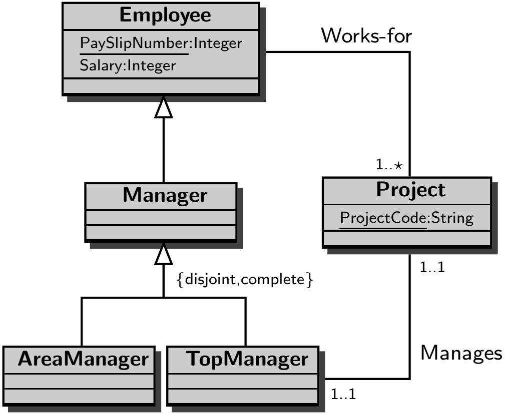
The role of a Conceptual Schema



The role of a Conceptual Schema

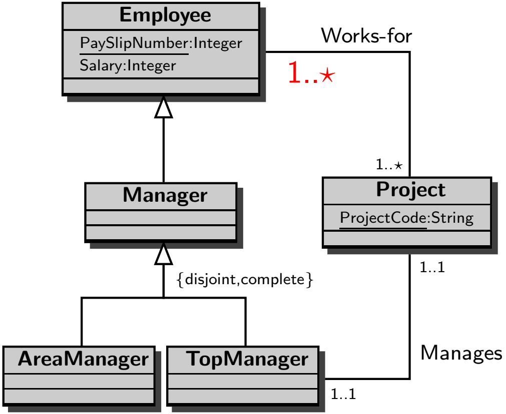


Reasoning with Conceptual Schemas



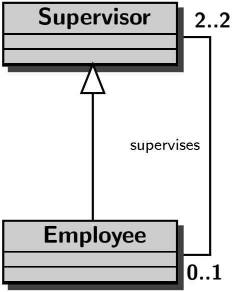
- Managers do not work for a project (she/he just manages it):
  $\forall x. \texttt{Manager}(x) \rightarrow \forall y. \neg \texttt{WORKS-FOR}(x, y)$

Reasoning with Conceptual Schemas

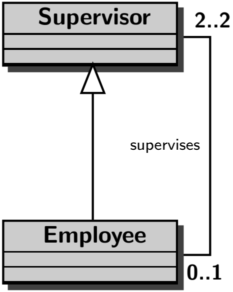


- Managers do not work for a project (she/he just manages it):
  $\forall x. \texttt{Manager}(x) \rightarrow \forall y. \neg \texttt{WORKS-FOR}(x, y)$
- If the minimum cardinality for the participation of employees to the *works-for* relationship is increased, then ...

Infinite Domain: the democratic company



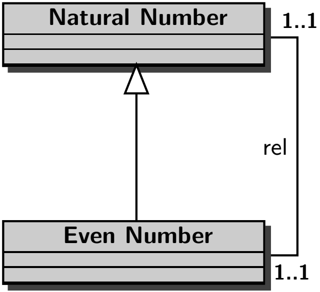
Infinite Domain: the democratic company



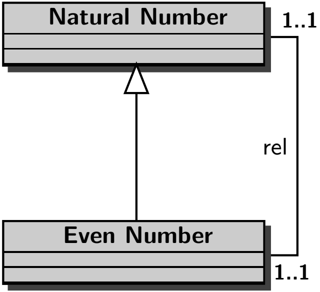
implies
"the classes Employee and Supervisor necessarily contain an infinite number of instances".

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

Bijection: how many numbers

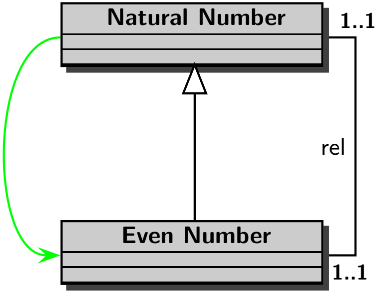


Bijection: how many numbers



implies
"the classes Natural Number and Even Number contain the same number of instances".

Bijection: 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

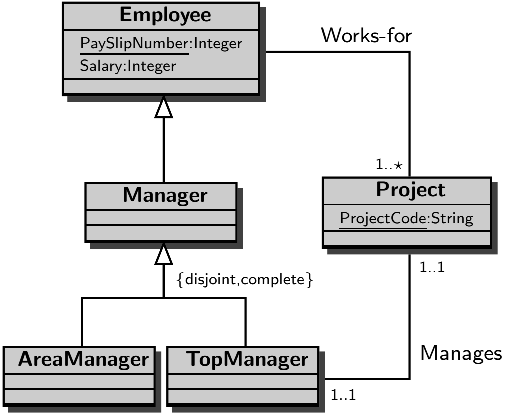
- Logic and Conceptual Modelling
- Description Logics for Conceptual Modelling
- Queries via a Conceptual Schema

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., DAML+OIL and 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., DAML+OIL and 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.

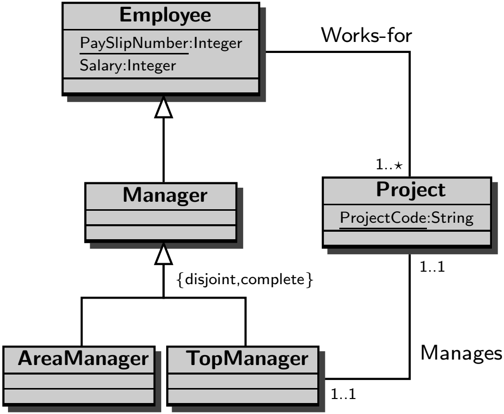


$$
\begin{array}{lcl}
\texttt{Works-for} & \sqsubseteq & \texttt{emp}/2 : \texttt{Employee} \sqcap \texttt{act}/2 : \texttt{Project} \\
\texttt{Manages} & \sqsubseteq & \texttt{man}/2 : \texttt{TopManager} \sqcap \texttt{prj}/2 : \texttt{Project} \\
\texttt{Employee} & \sqsubseteq & \exists^{=1}[\texttt{worker}](\texttt{PaySlipNumber}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

Set-based Constraints

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

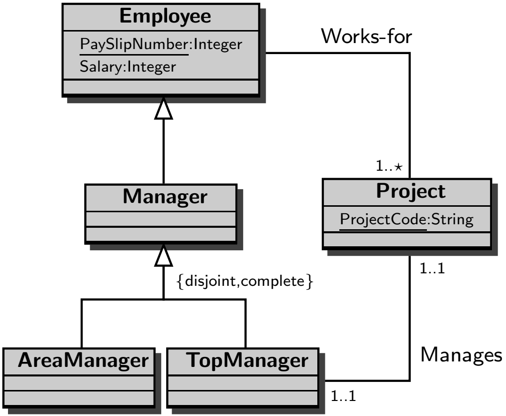
Deducing constraints



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

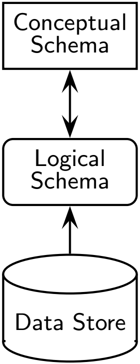
i•com: **Intelligent Conceptual Modelling**

- 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.

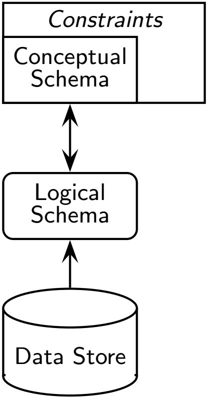
Summary

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

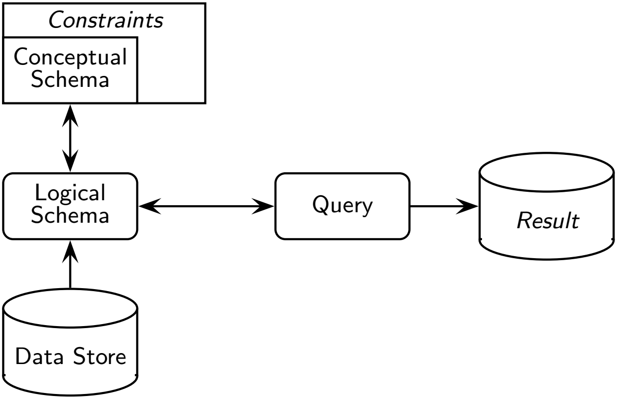
The role of a Conceptual Schema – revisited



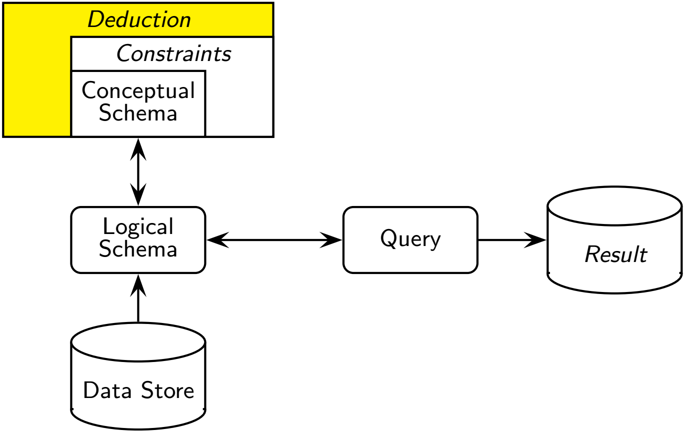
The role of a Conceptual Schema – revisited



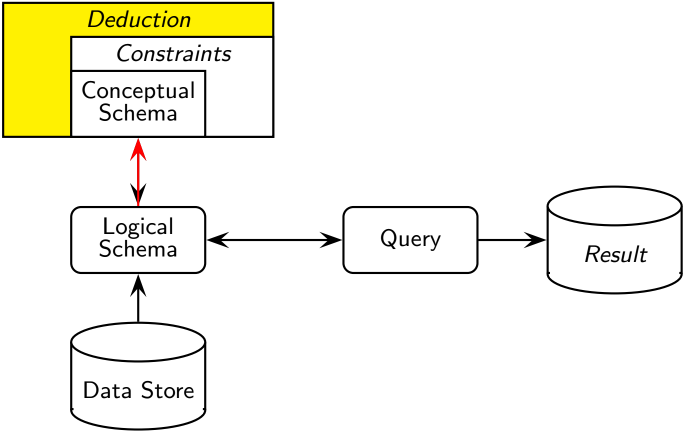
The role of a Conceptual Schema – revisited



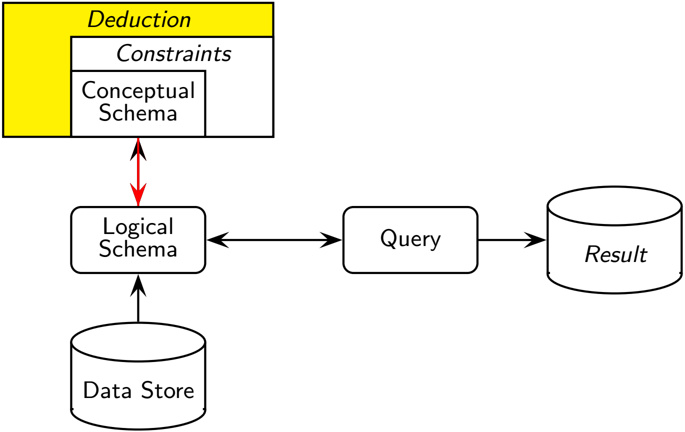
The role of a Conceptual Schema – revisited



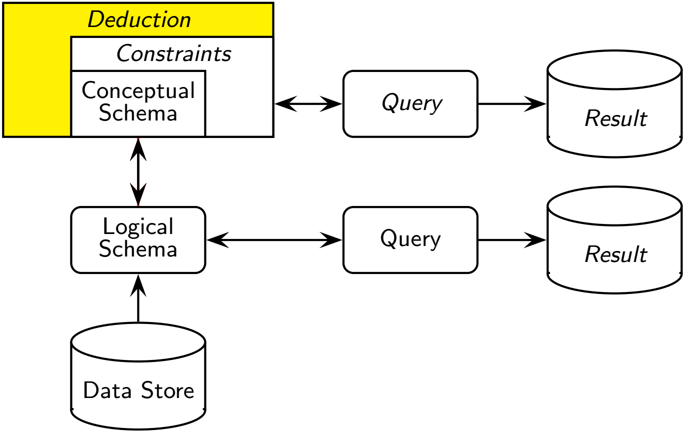
The role of a Conceptual Schema – revisited



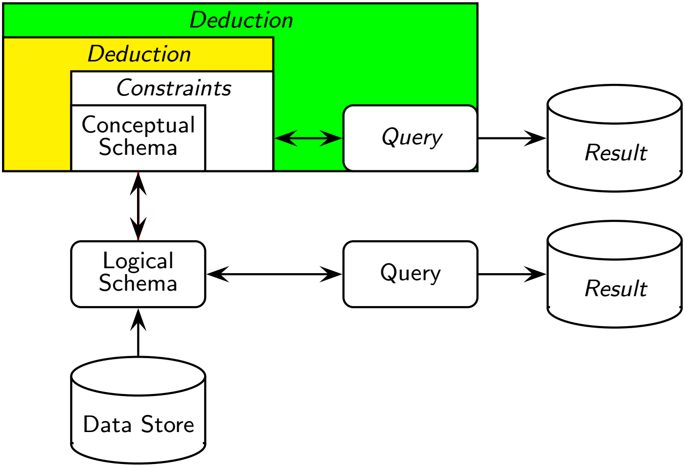
The role of a Conceptual Schema – revisited



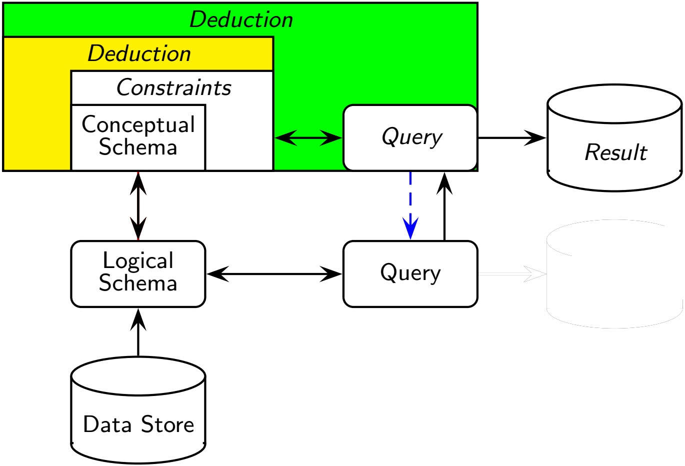
The role of a Conceptual Schema – revisited



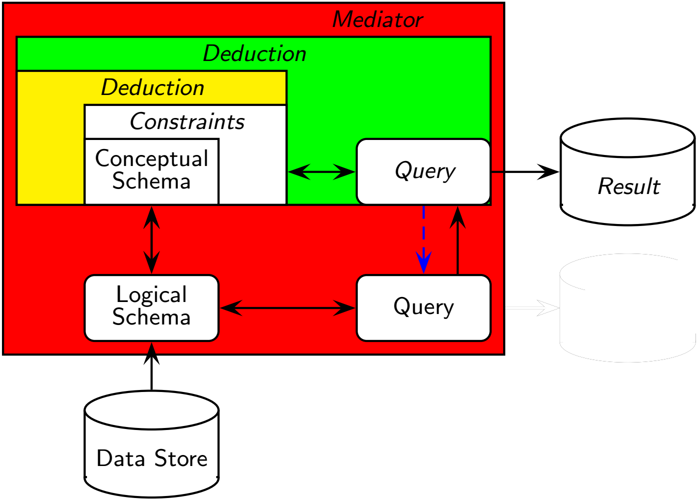
The role of a Conceptual Schema – revisited



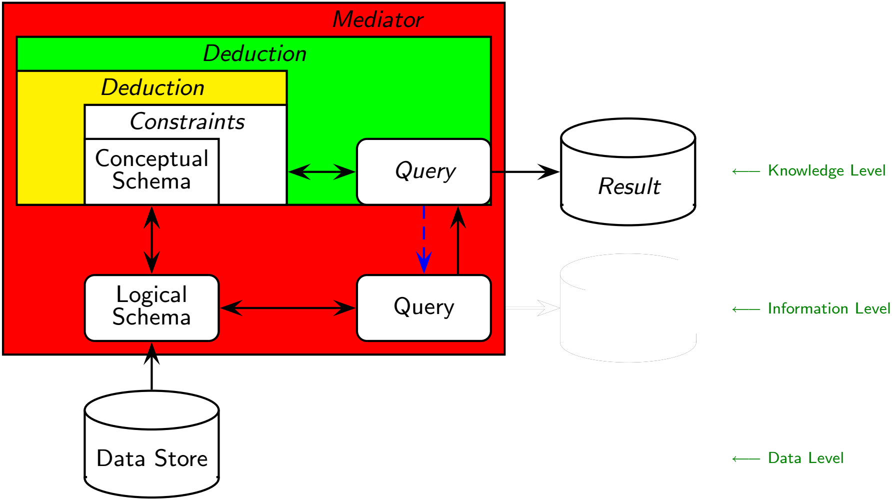
The role of a Conceptual Schema – revisited



The role of a Conceptual Schema – revisited



The role of a Conceptual Schema – revisited



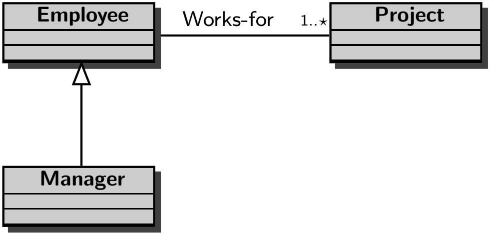
Queries via Conceptual Schemas: the DB assumption

- 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

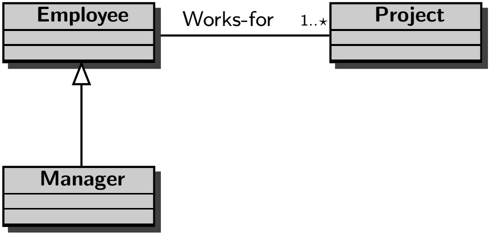
Queries via Conceptual Schemas: the DB assumption

- 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

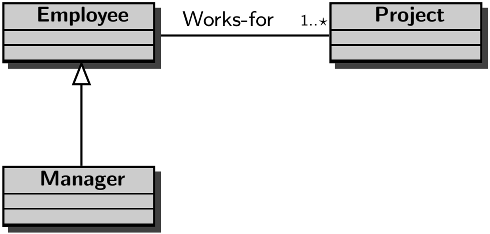


Example with DB assumption



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

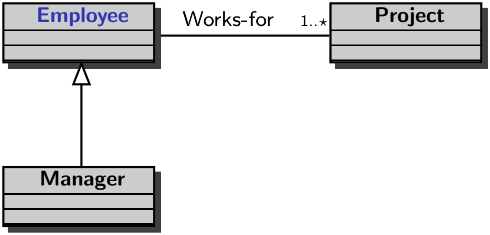
Weakening the DB assumption

- 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: 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

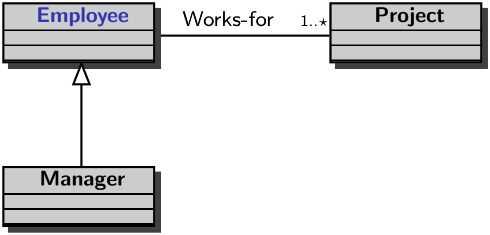


Manager = { John, Paul }

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

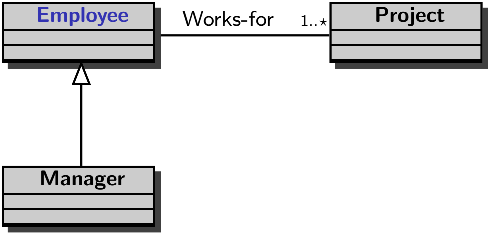
Project = { Prj-A, Prj-B }

Example with partial DB assumption




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

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

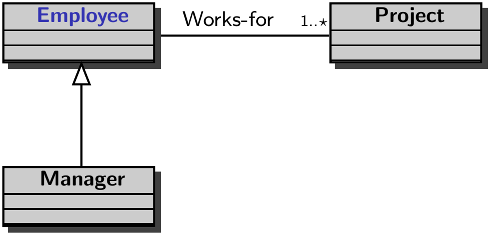
Example with partial DB assumption



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

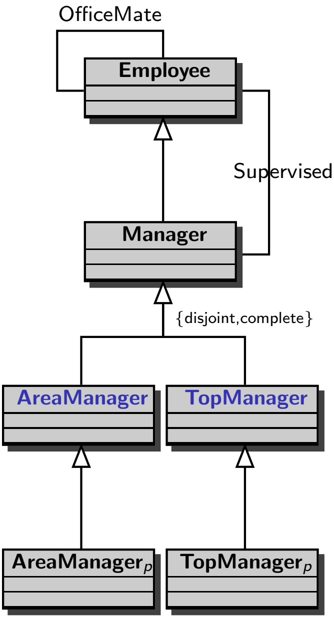
Q(X) :- Employee(X)
$\Longrightarrow$ { John, Paul, Mary }

Example with partial DB assumption

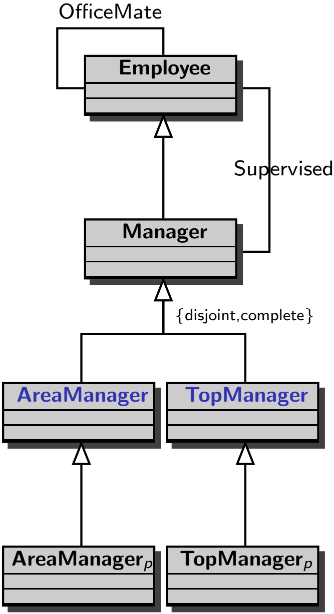



```
Manager = { John, Paul }
Works-for = { ⟨John,Prj-A⟩, ⟨Mary,Prj-B⟩ }
Project = { Prj-A, Prj-B }

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

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

Andrea's Example

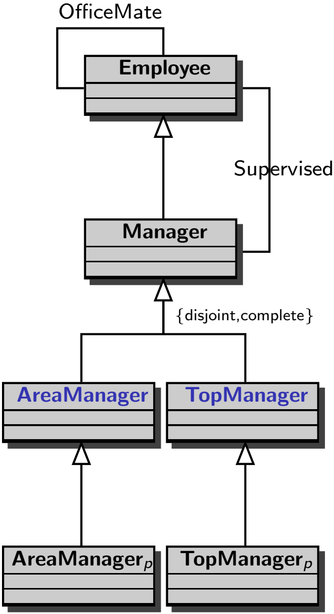


Andrea's Example

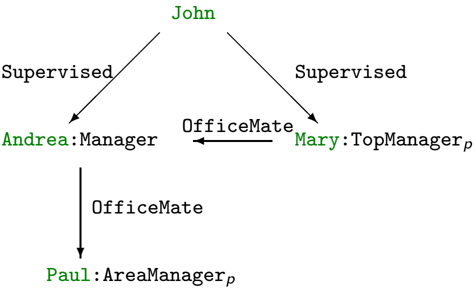



```
Employee = { Andrea, Paul, Mary, John }
Manager = { Andrea, Paul, Mary}
AreaManager_p = { Paul }
TopManager_p = { Mary }
Supervised = { ⟨John,Andrea⟩, ⟨John,Mary⟩ }
OfficeMate = { ⟨Mary,Andrea⟩, ⟨Andrea,Paul⟩ }
```

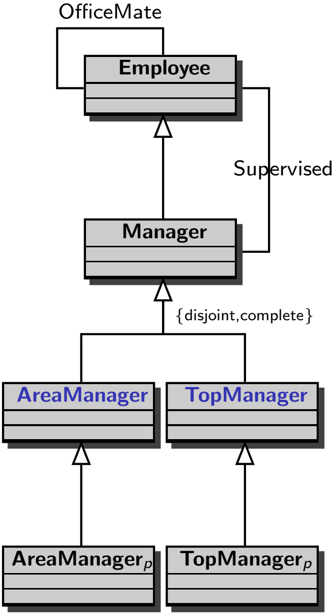
Andrea's Example



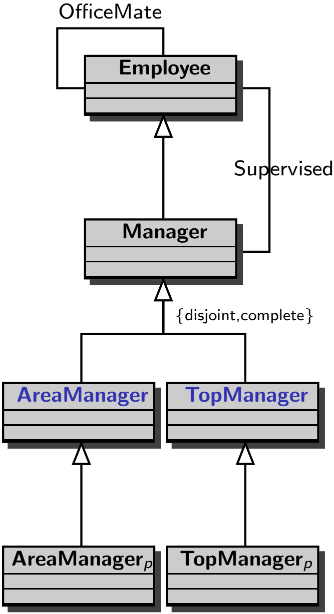
Employee = { Andrea, Paul, Mary, John }
Manager = { Andrea, Paul, Mary}
AreaManager$_p$ = { Paul }
TopManager$_p$ = { Mary }
Supervised = { ⟨John,Andrea⟩, ⟨John,Mary⟩ }
OfficeMate = { ⟨Mary,Andrea⟩, ⟨Andrea,Paul⟩ }

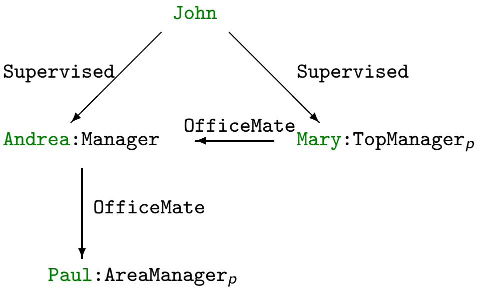


Andrea's Example (cont.)

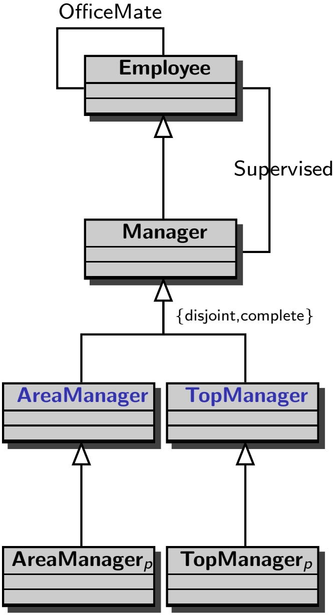


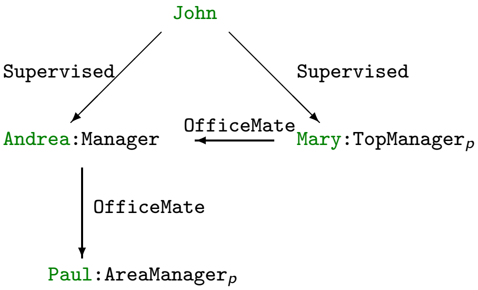
Andrea's Example (cont.)





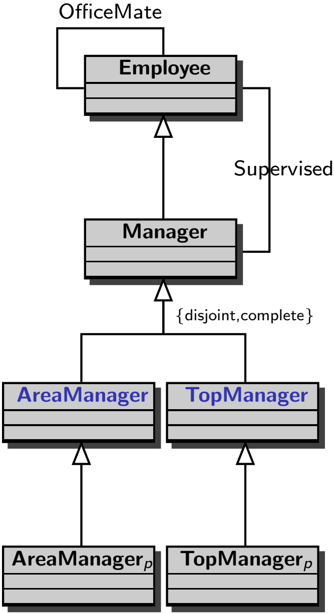
Andrea’s Example (cont.)

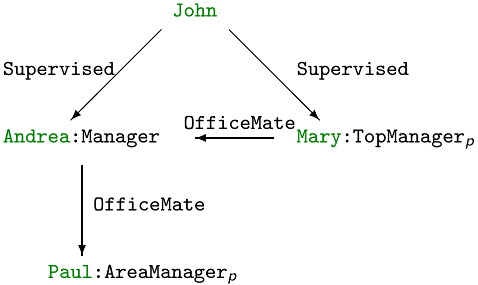





```
Q(X) :- Supervised(X,Y), TopManager(Y),
        Officemate(Y,Z), AreaManager(Z)
```

Andrea's Example (cont.)





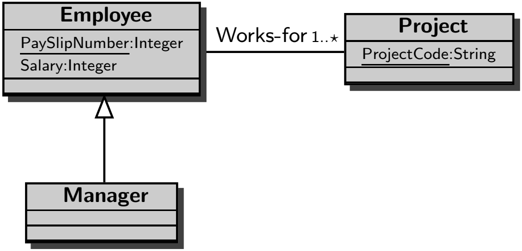

```
Q(X) :- Supervised(X,Y), TopManager(Y),
        Officemate(Y,Z), AreaManager(Z)
```

$\Longrightarrow$ { John }

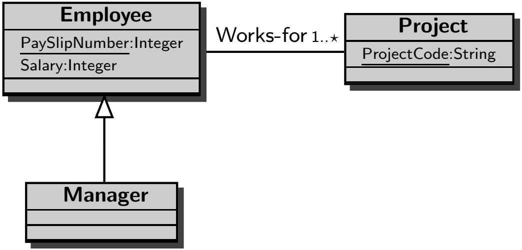
View based Query Processing

- 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.

Sound GAV mapping

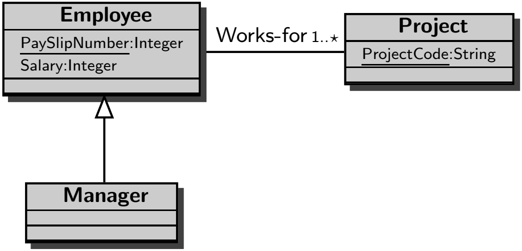


Sound GAV mapping




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

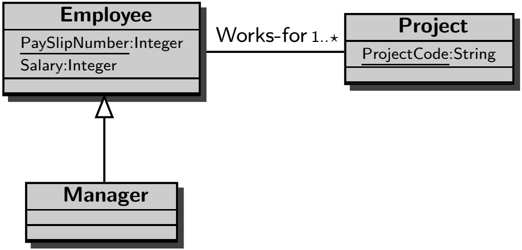
Sound GAV mapping



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

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

Sound GAV mapping

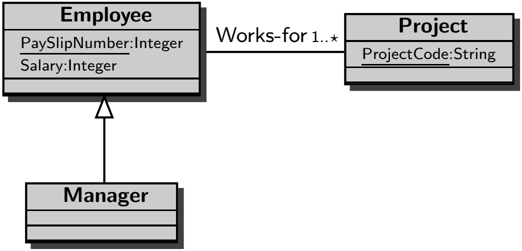



```
1-Employee(PaySlipNumber,Salary,ManagerP)
2-Works-for(PaySlipNumber,ProjectCode)
```

```
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

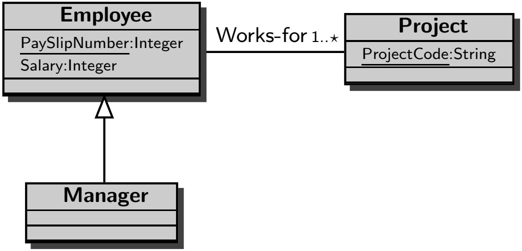



```
1-Employee(PaySlipNumber,Salary,ManagerP)
2-Works-for(PaySlipNumber,ProjectCode)
```

```
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



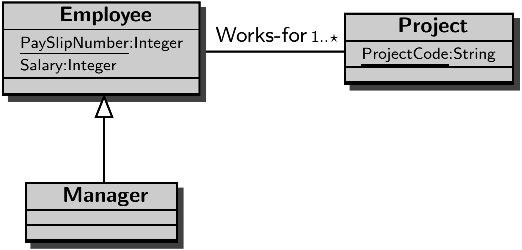

```
1-Employee(PaySlipNumber,Salary,ManagerP)
2-Works-for(PaySlipNumber,ProjectCode)
```

```
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)
```

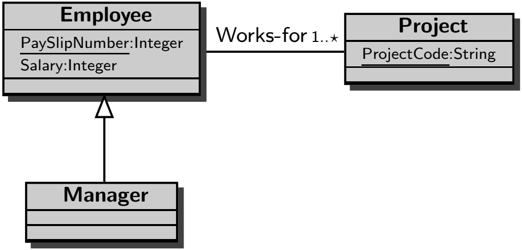
```
Q(X) :- Employee(X)
⟹        Q'(X) :- 1-Employee(X,Y,Z) ∪ 2-Works-for(X,W)
```

← not coming from unfolding!

Sound LAV mapping

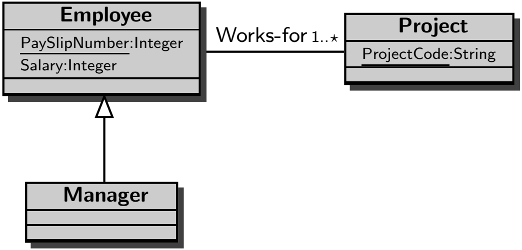


Sound LAV mapping



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

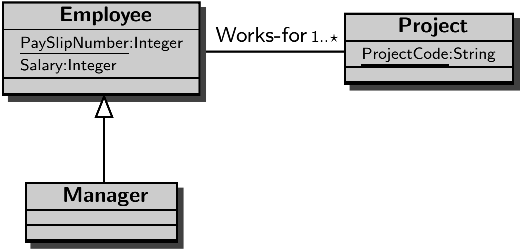
Sound LAV mapping




```
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

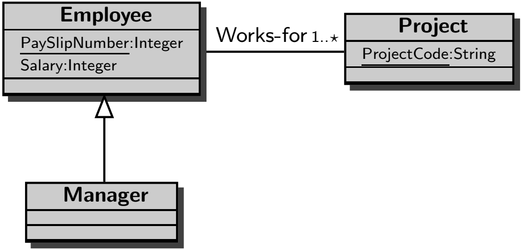


1-Employee(PaySlipNumber,Salary,ManagerP)
2-Works-for(PaySlipNumber,ProjectCode)

```
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



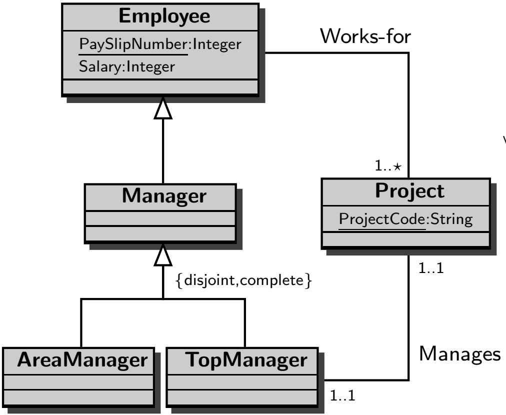
1-Employee(PaySlipNumber,Salary,ManagerP)
2-Works-for(PaySlipNumber,ProjectCode)

```
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)
$\Longrightarrow$ Q'(X) :- 1-Employee(X,Y,true), 2-Works-for(X,Z)

Reasoning over queries

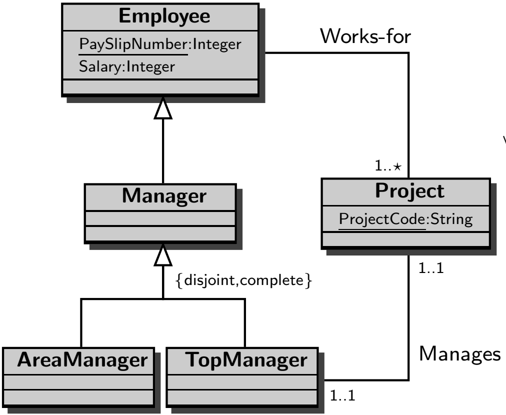
`Q(X,Y) :- Employee(X), Works-for(X,Y), Manages(X,Y)`



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

Reasoning over queries

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



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

$\rightsquigarrow$ 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!