(Description) Logics for

Information Modelling and Access

- or -

How to Use an Ontology

Enrico Franconi

franconi@inf.unibz.it

http://www.inf.unibz.it/˜franconi

Faculty of Computer Science, Free University of Bozen-Bolzano

Summary

- What is an Ontology
- Description Logics for Conceptual Modelling
- Queries with an Ontology

What is an Ontology

- An ontology is a formal conceptualisation of the world: 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.

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.

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.

Ontology languages 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 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., having only concepts and taxonomies), frame-based (having only concepts and properties), or logic-based (e.g. Ontolingua, DAML+OIL, OWL).

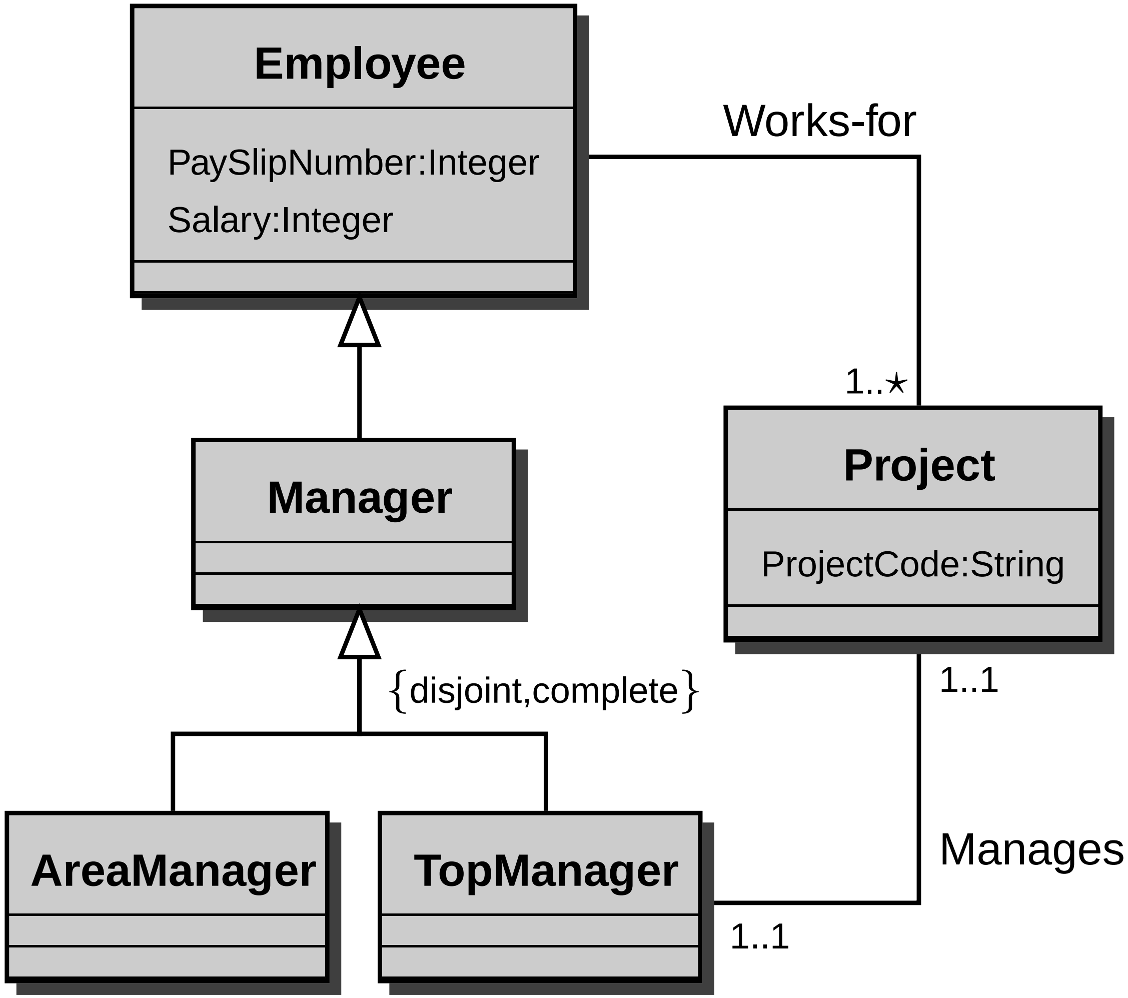
Ontology languages and Conceptual Data Models

- 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., having only concepts and taxonomies), frame-based (having only concepts and properties), or logic-based (e.g. Ontolingua, DAML+OIL, OWL).
- Ontology languages are typically expressed by means of diagrams.

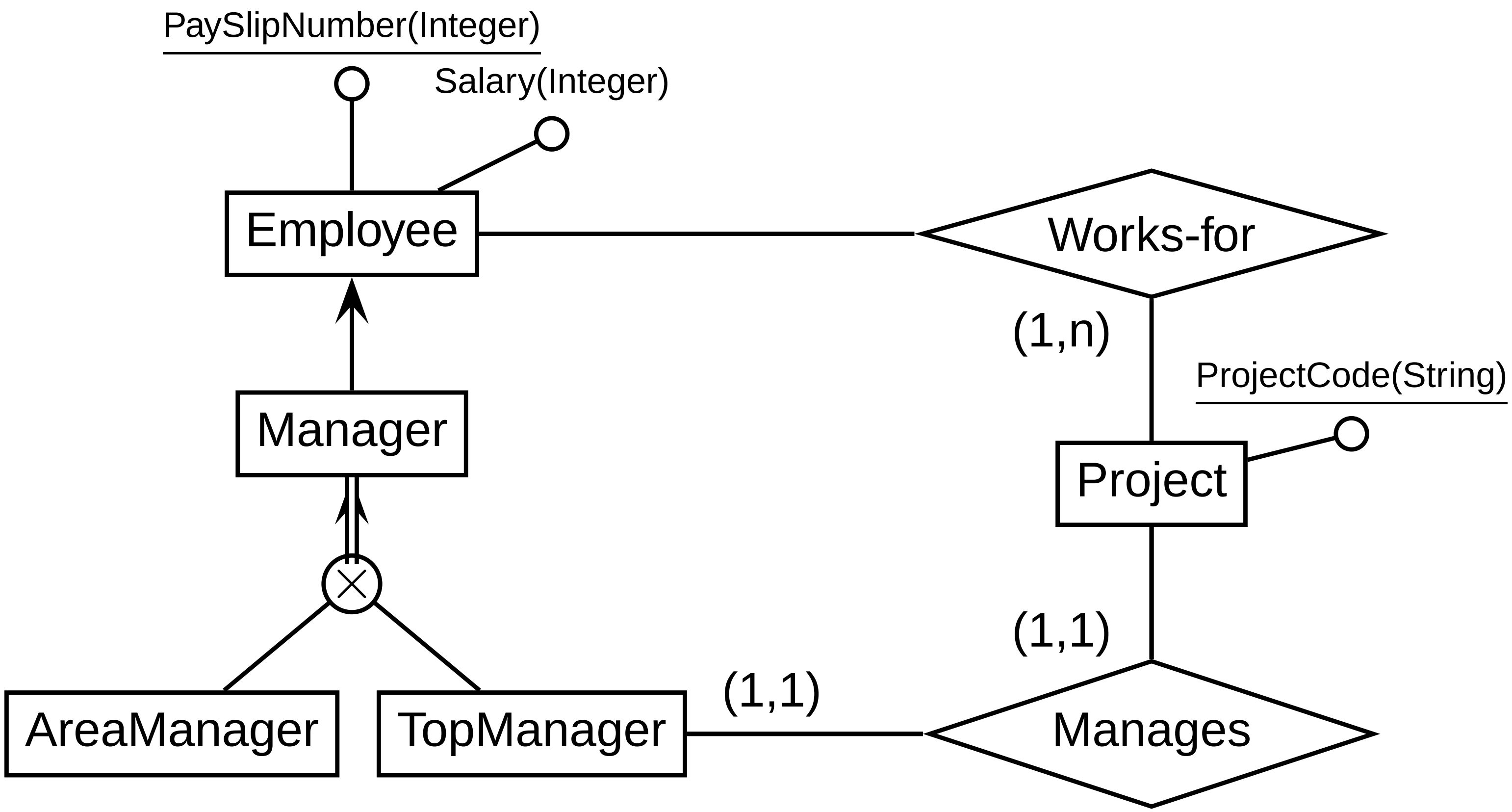
Ontology languages and Conceptual Data Models

- 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., having only concepts and taxonomies), frame-based (having only concepts and properties), or logic-based (e.g. Ontolingua, DAML+OIL, OWL).
- Ontology languages are typically expressed by means of diagrams.
- **Entity-Relationship** schemas and **UML** class diagrams can be considered as ontologies.

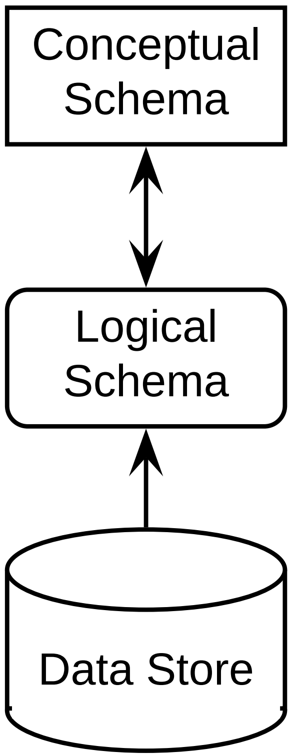
UML Class Diagram



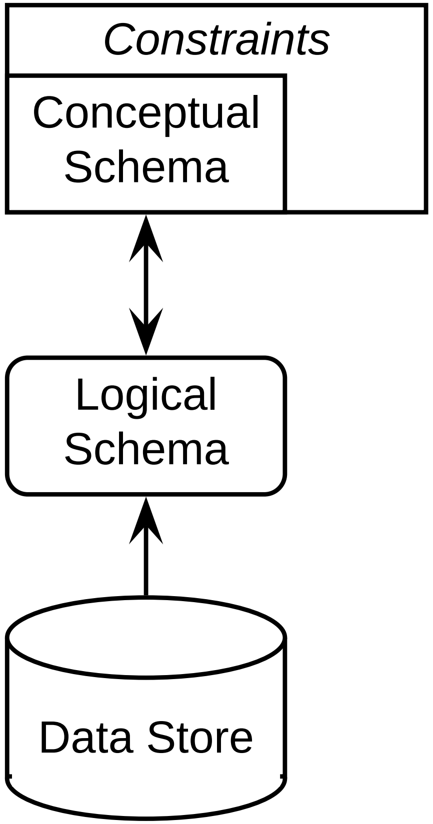
Entity-Relationship Schema



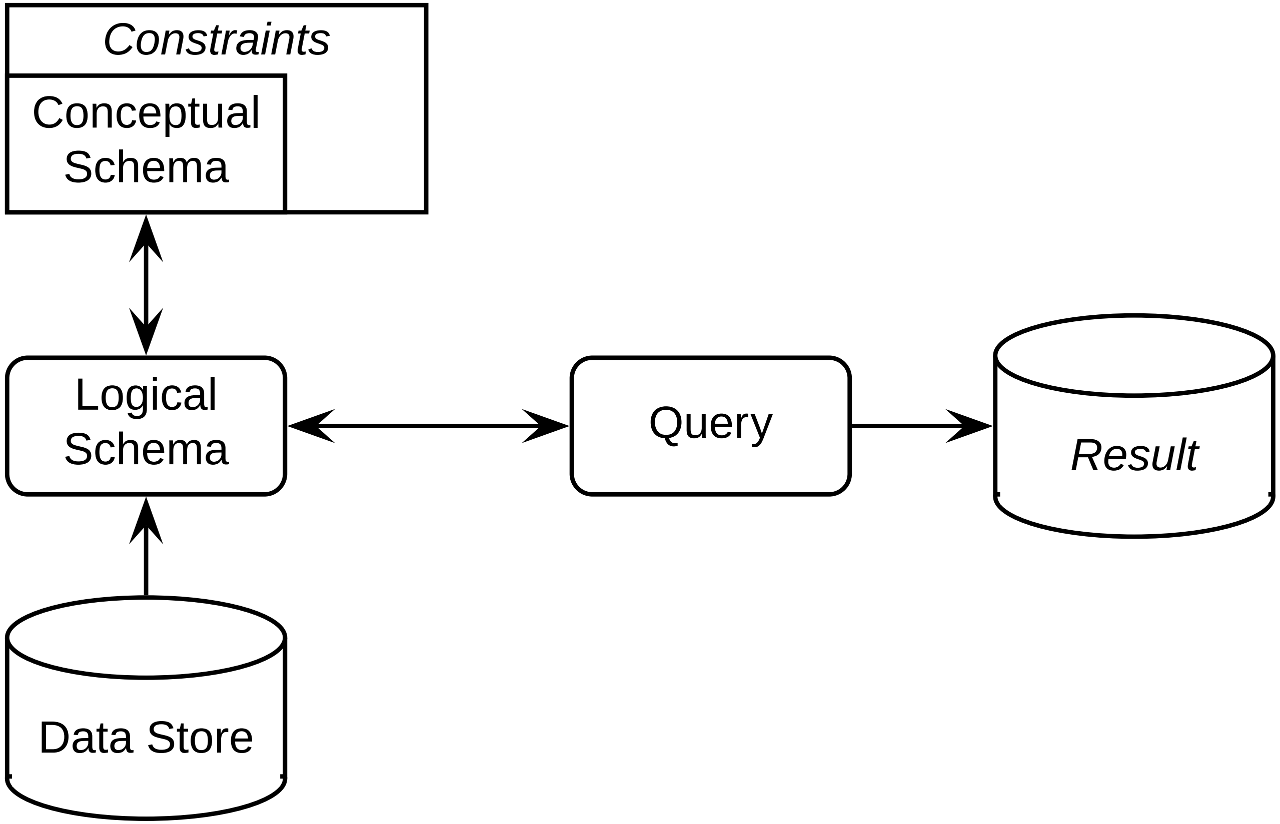
The role of a Conceptual Schema



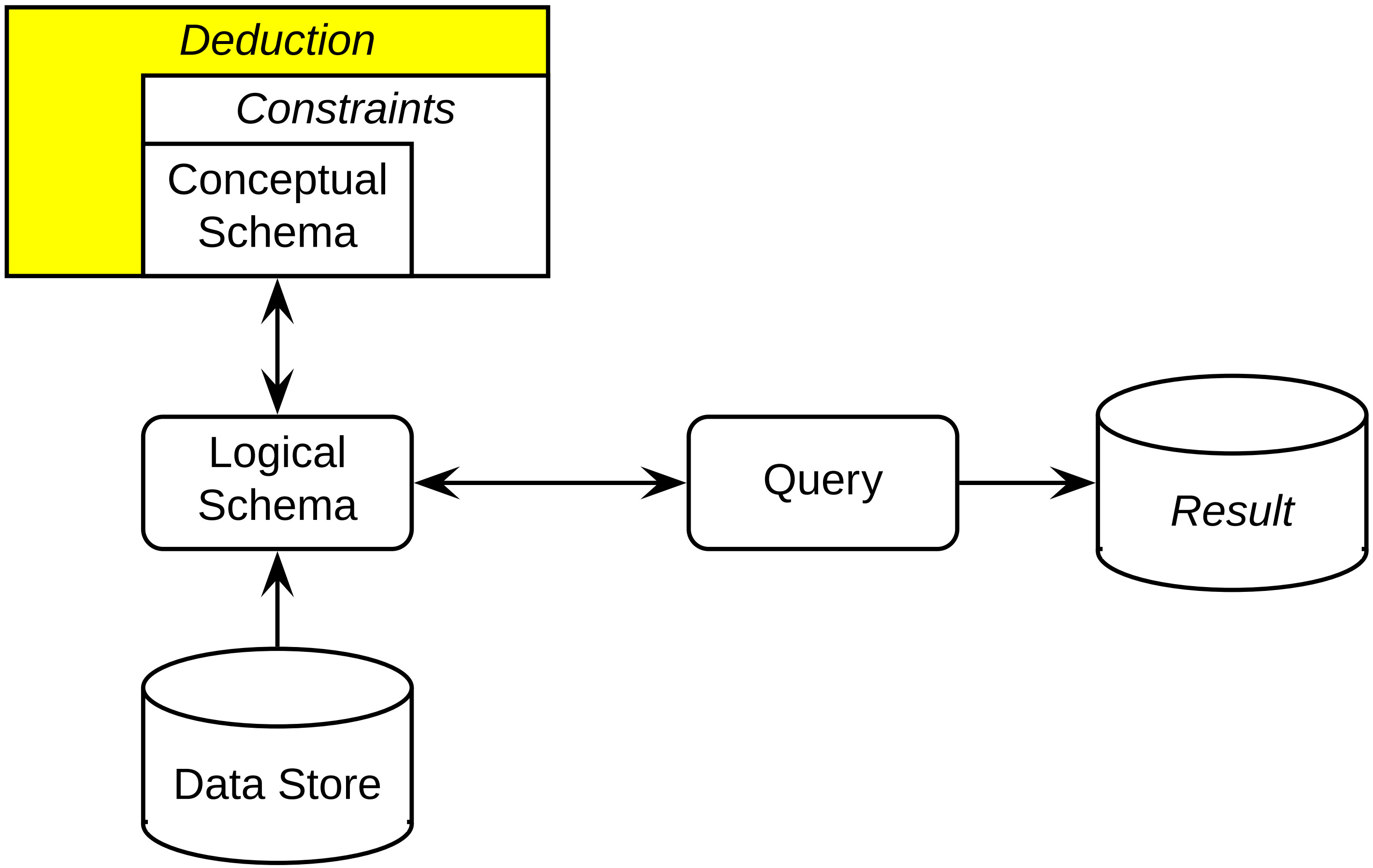
The role of a Conceptual Schema



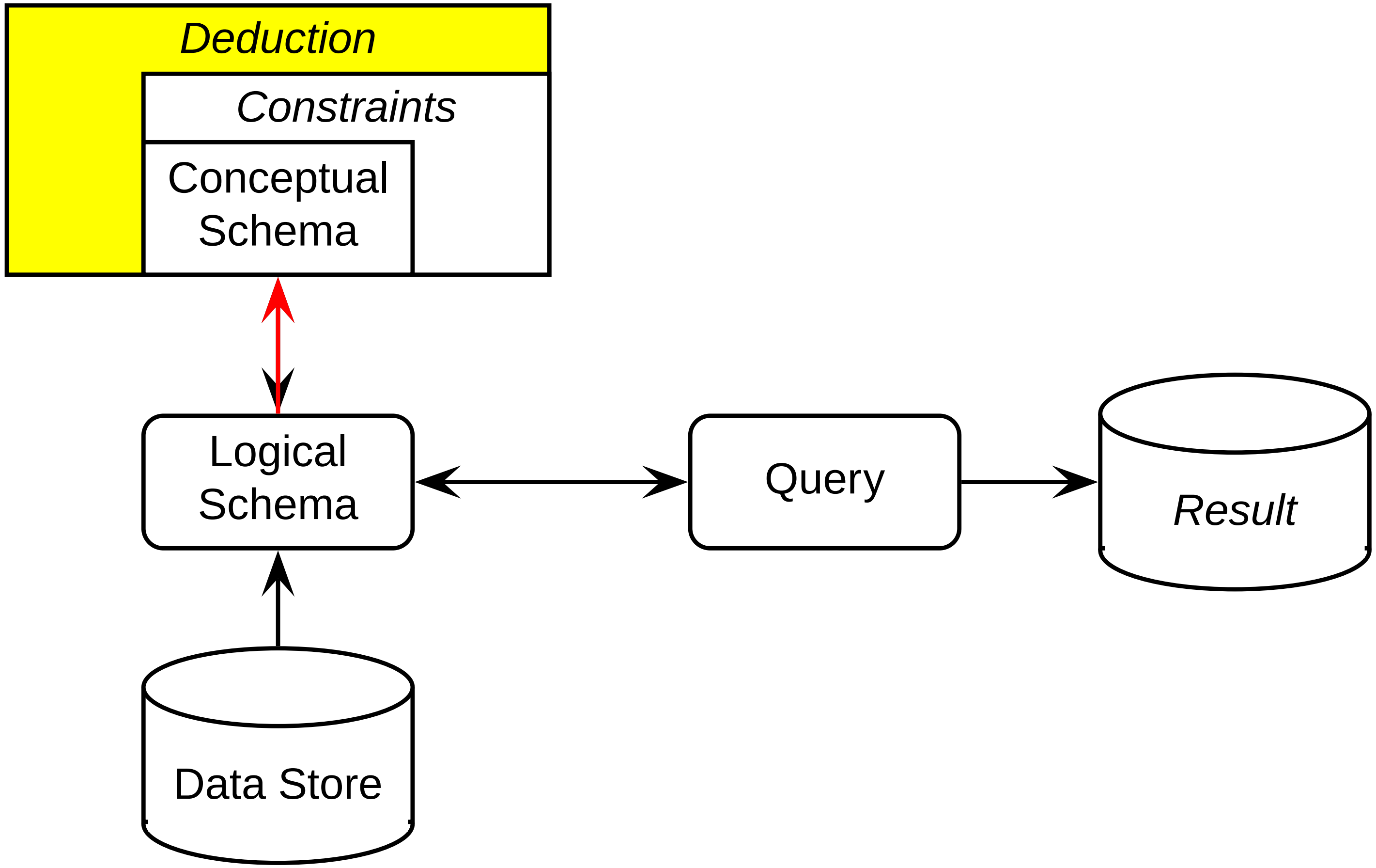
The role of a Conceptual Schema



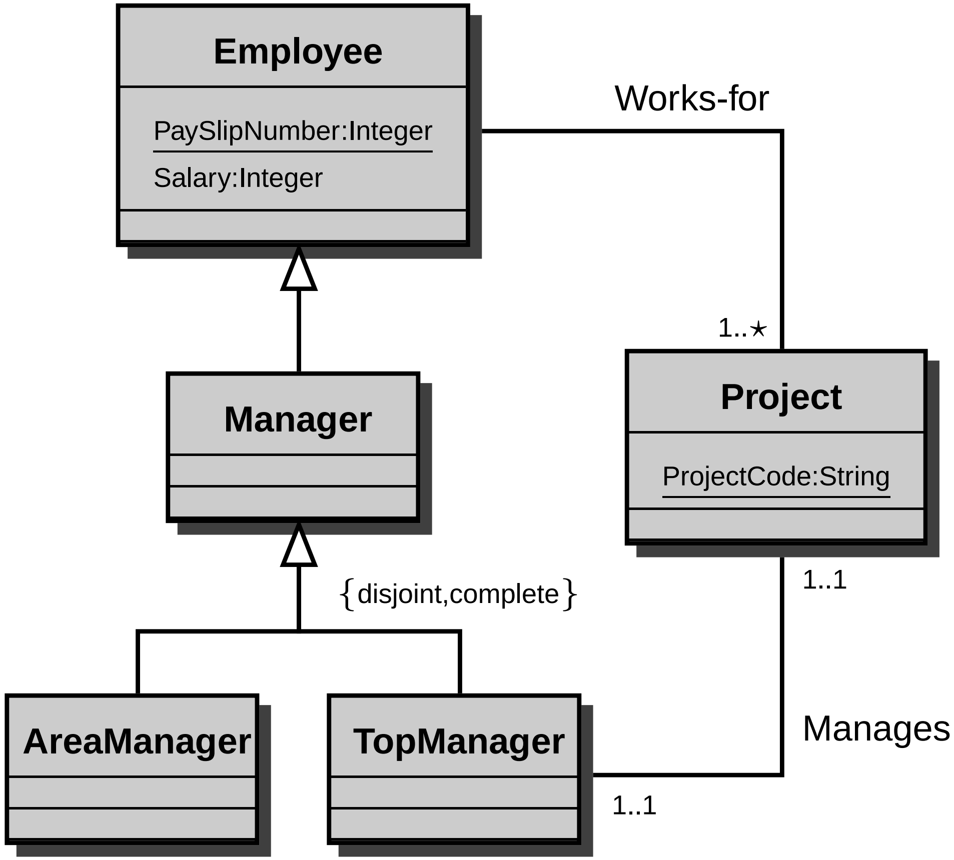
The role of a Conceptual Schema



The role of a Conceptual Schema



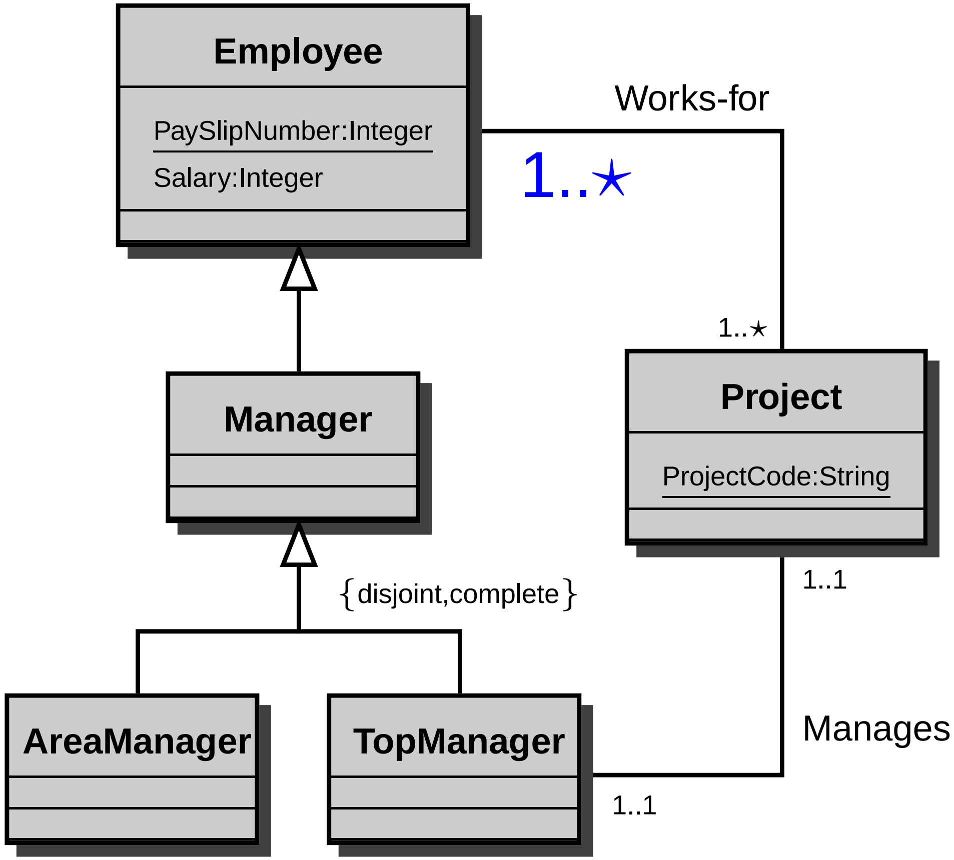
Reasoning with Ontologies



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

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

Reasoning with Ontologies



- 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 . . .

Summary

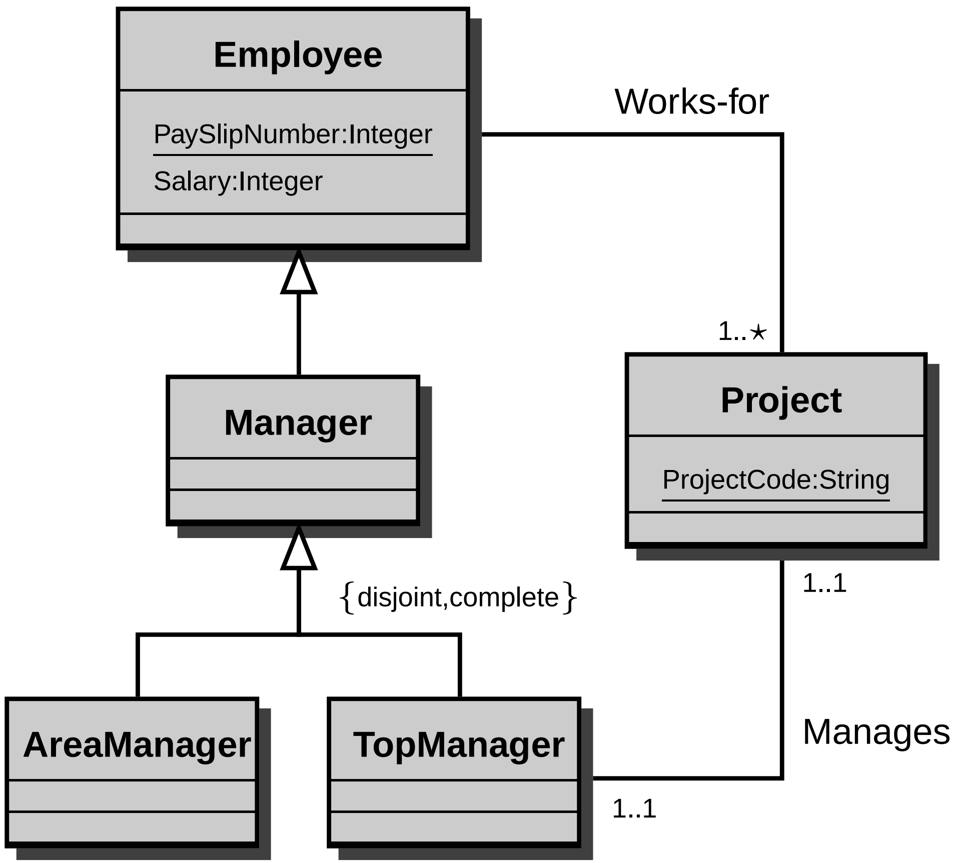
- Logic and Conceptual Modelling
- Description Logics for Conceptual Modelling
- Queries with an Ontology

Encoding ontologies 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)

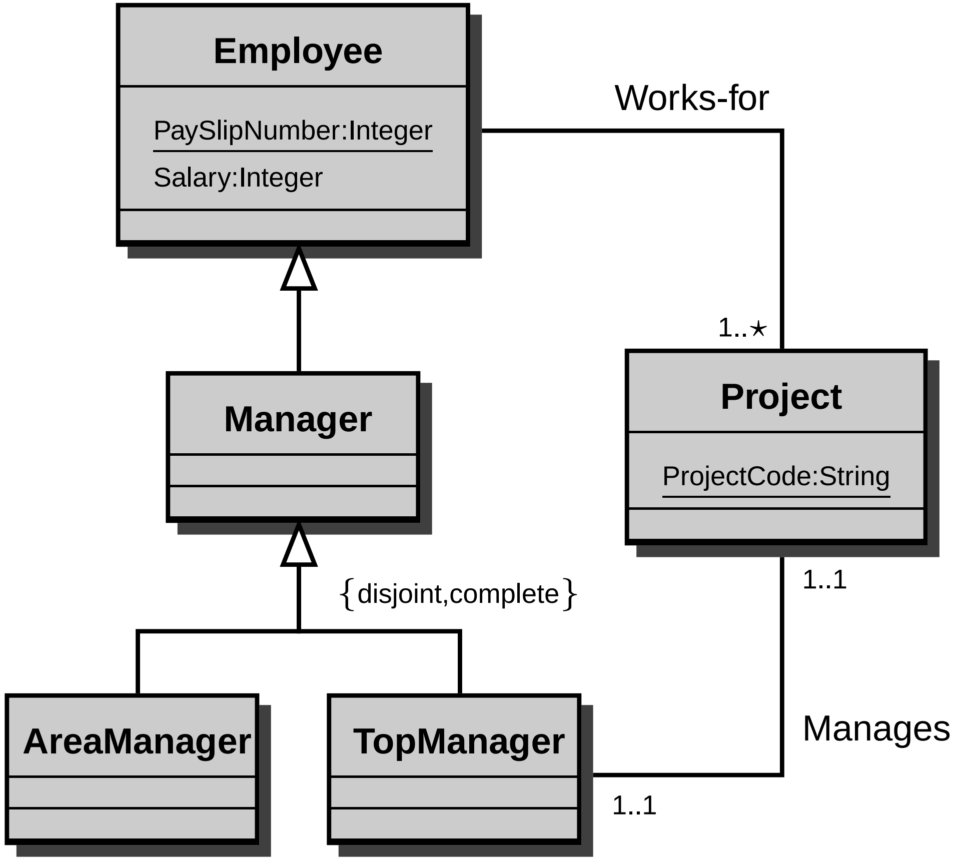
Encoding ontologies in Description Logics

- 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 an ontology 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 ontology, and vice-versa.



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

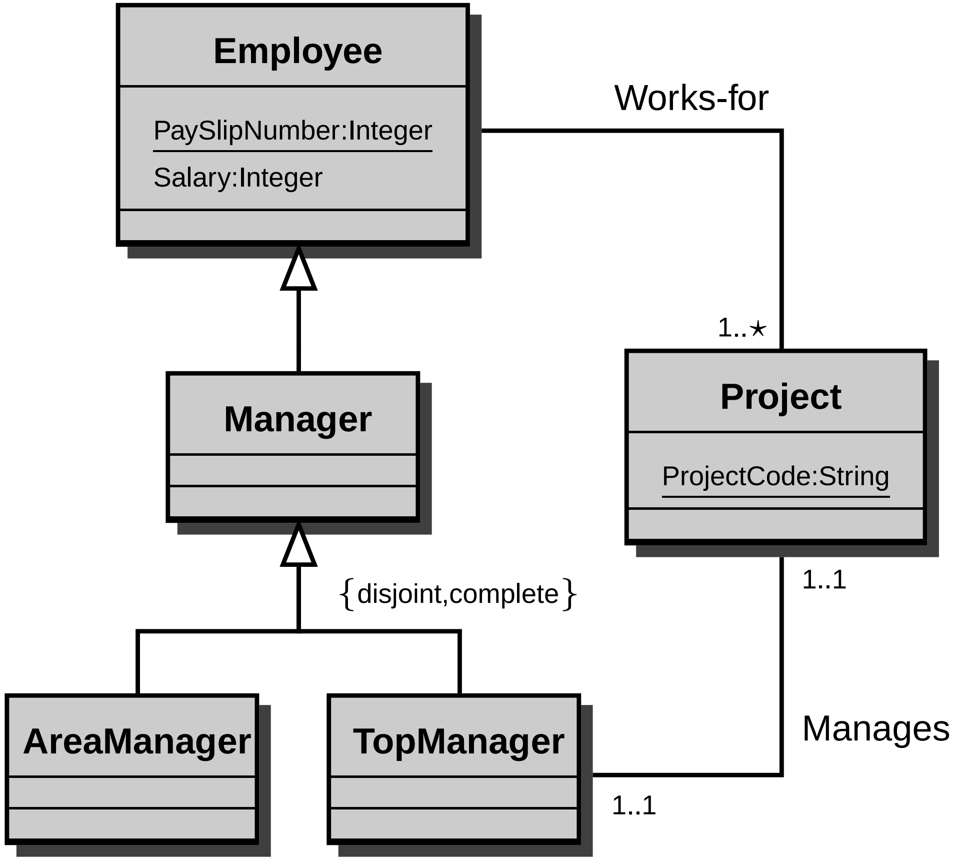
Deducing constraints



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

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

Deducing constraints



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

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

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

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

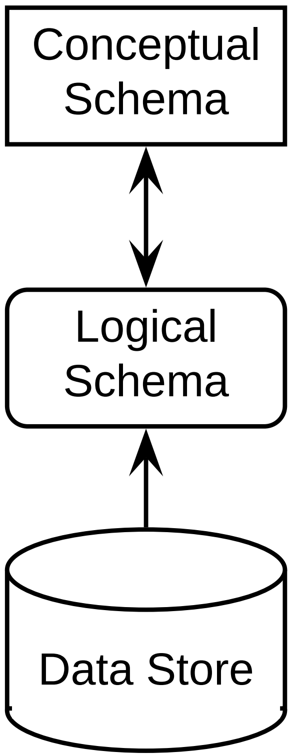
i●com: **Intelligent Conceptual Modelling tool**

- 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.

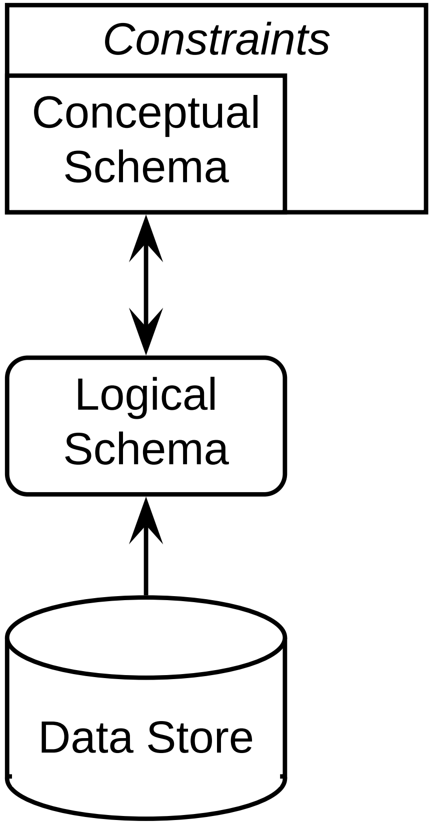
Summary

- Logic and Conceptual Modelling
- Description Logics for Conceptual Modelling
- $\boxed{\text{Queries with an Ontology}}$

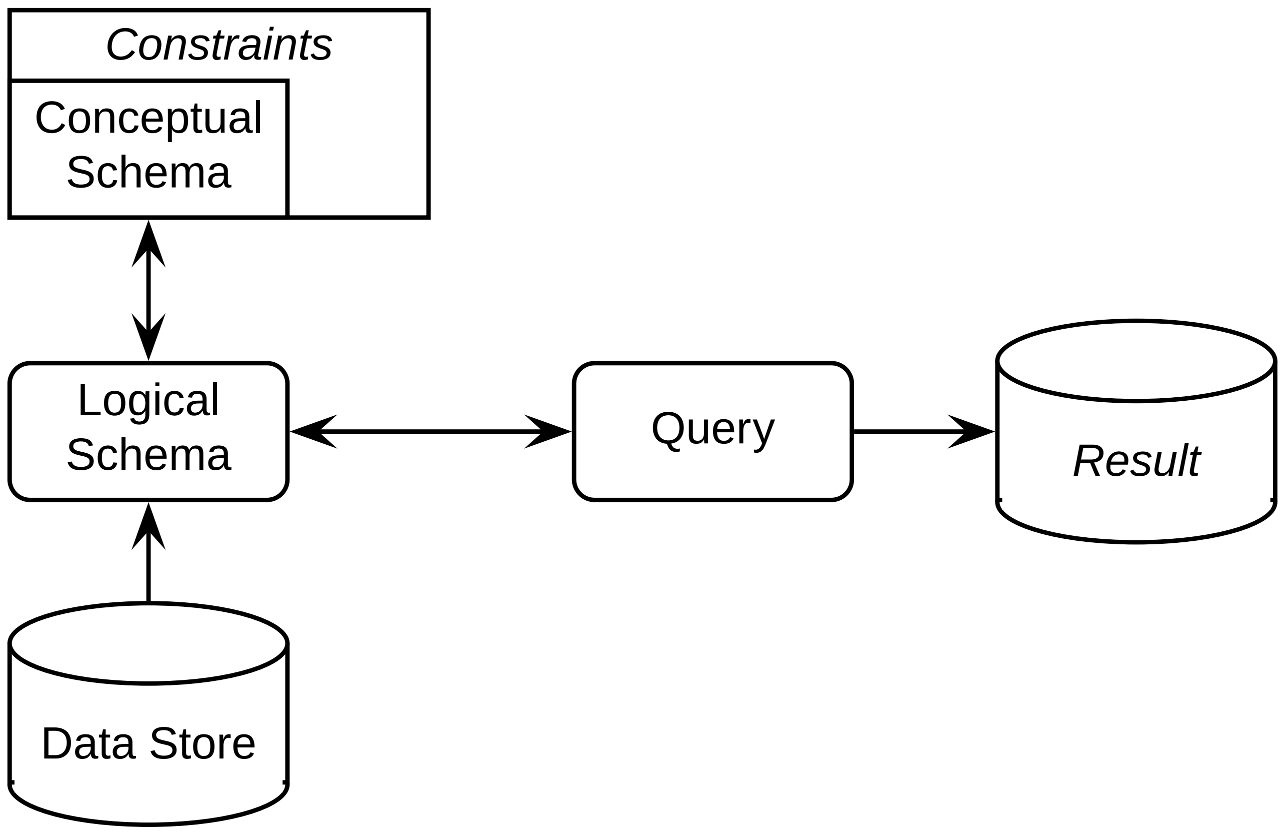
The role of a Conceptual Schema – revisited



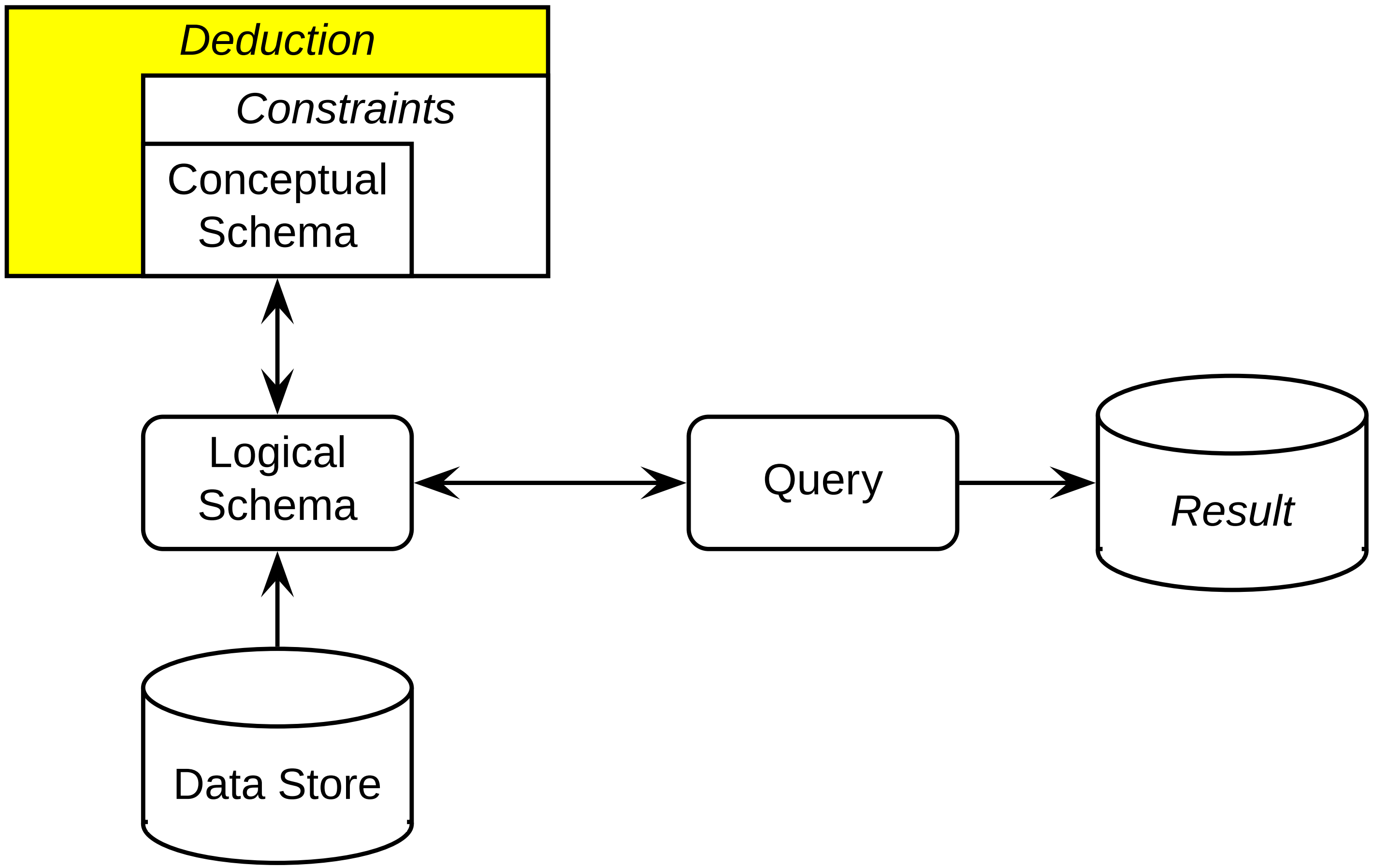
The role of a Conceptual Schema – revisited



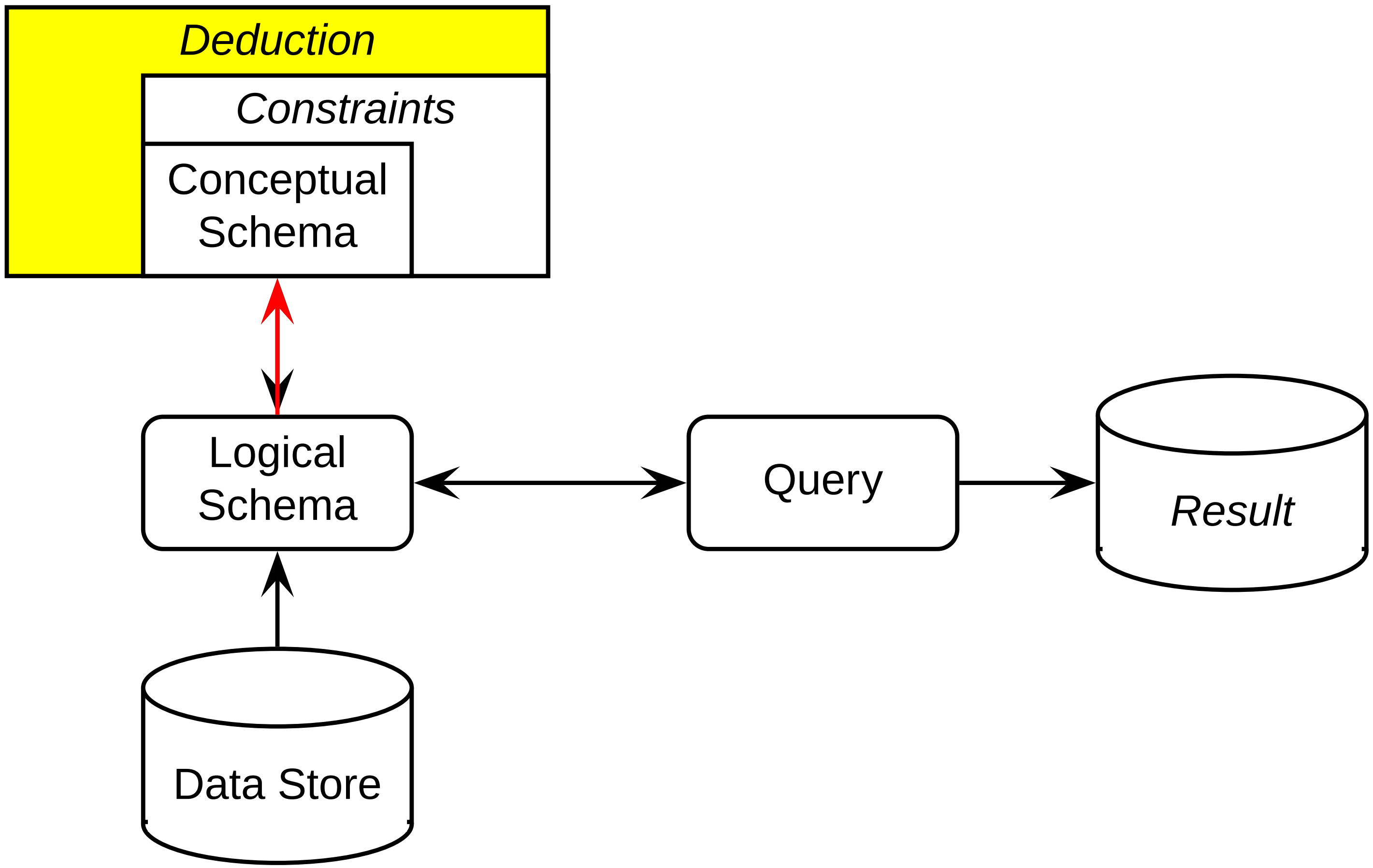
The role of a Conceptual Schema – revisited



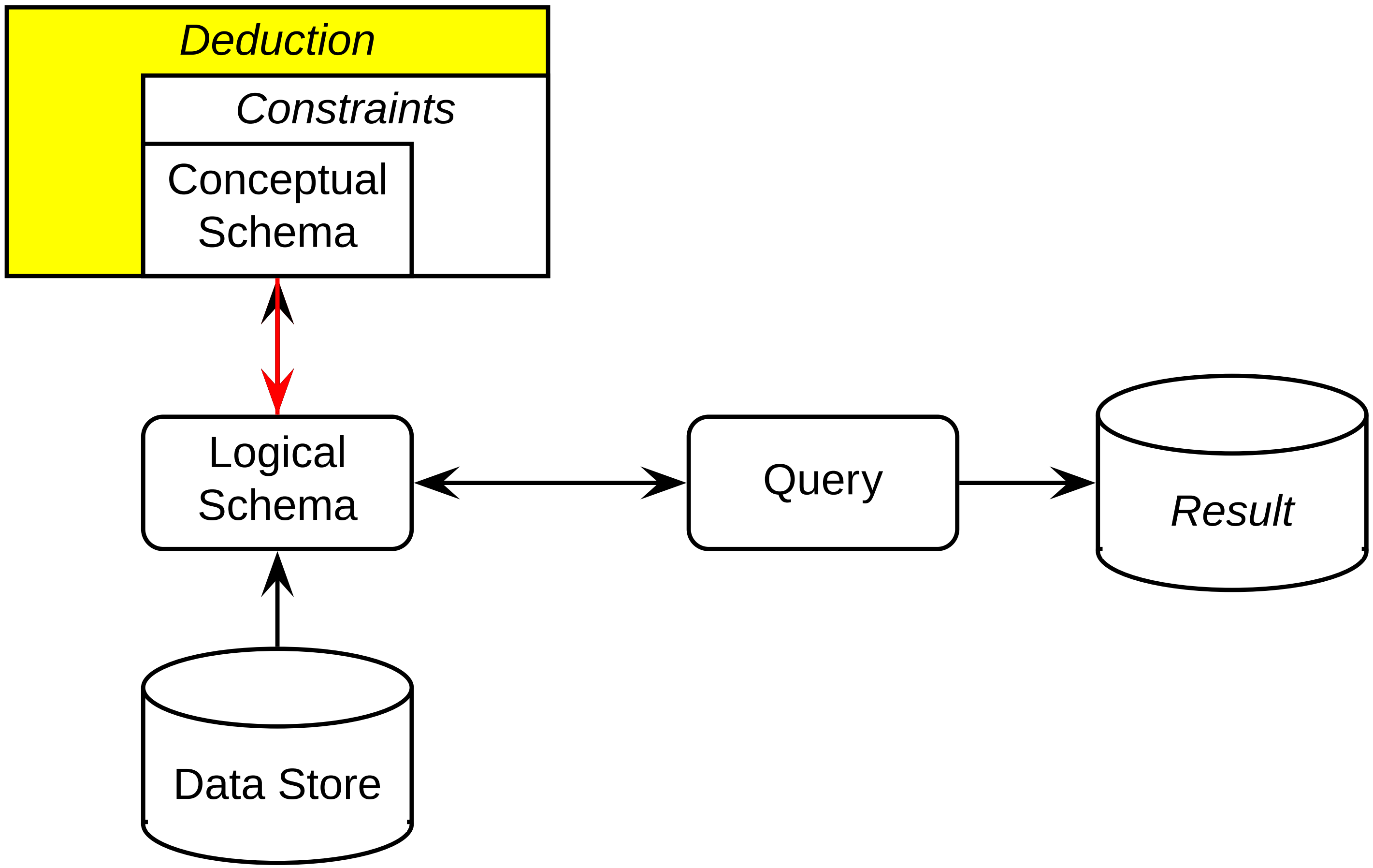
The role of a Conceptual Schema – revisited



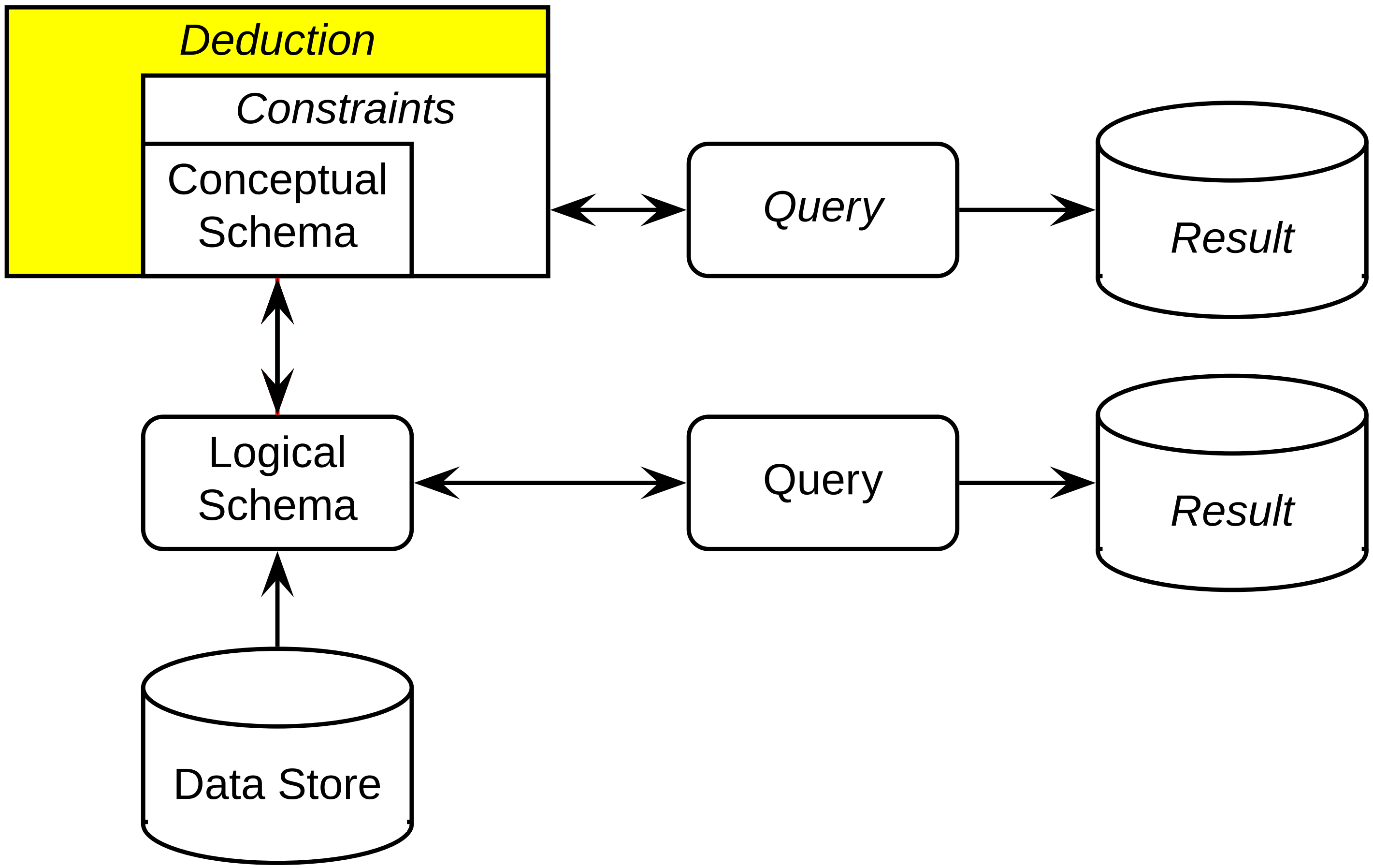
The role of a Conceptual Schema – revisited



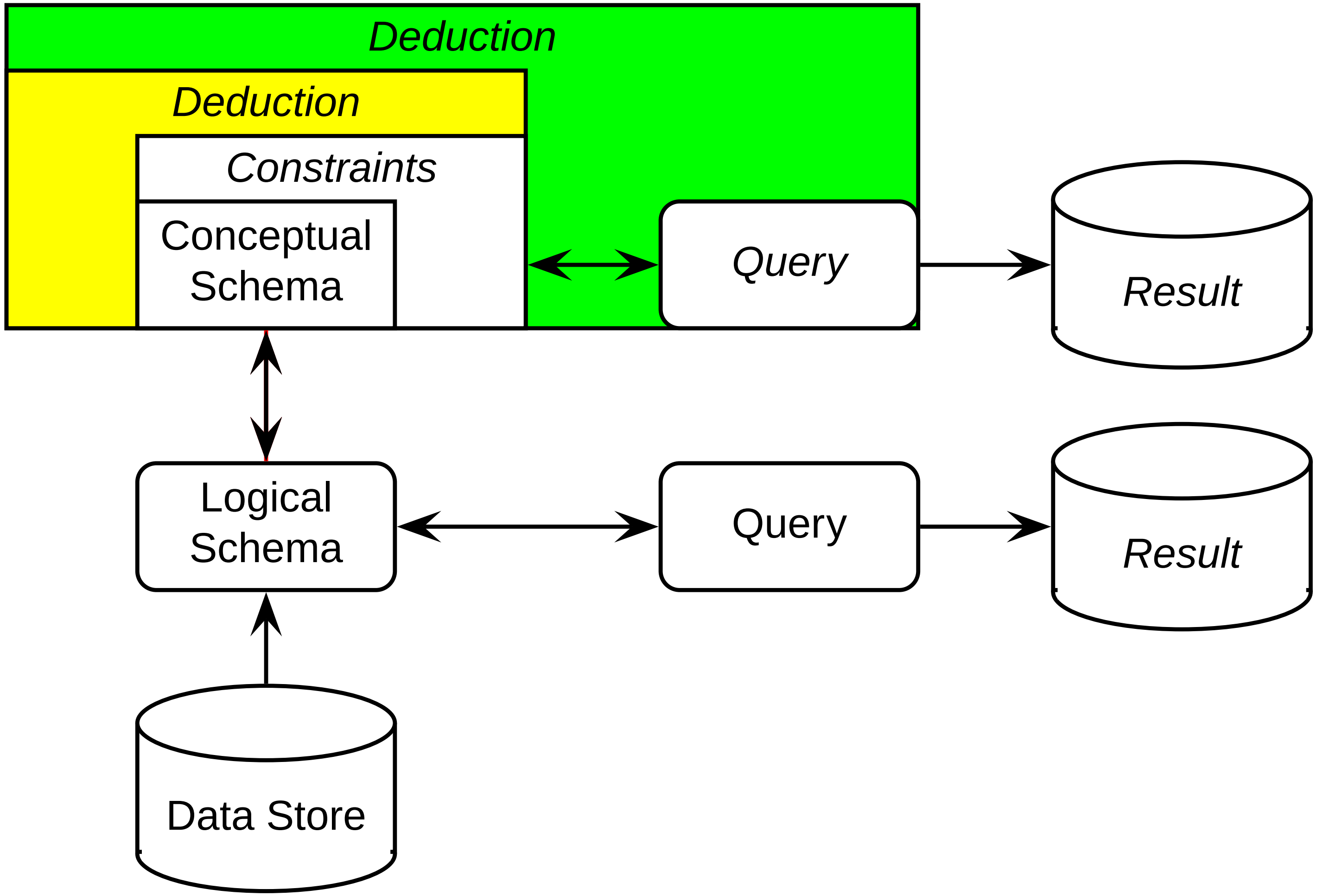
The role of a Conceptual Schema – revisited



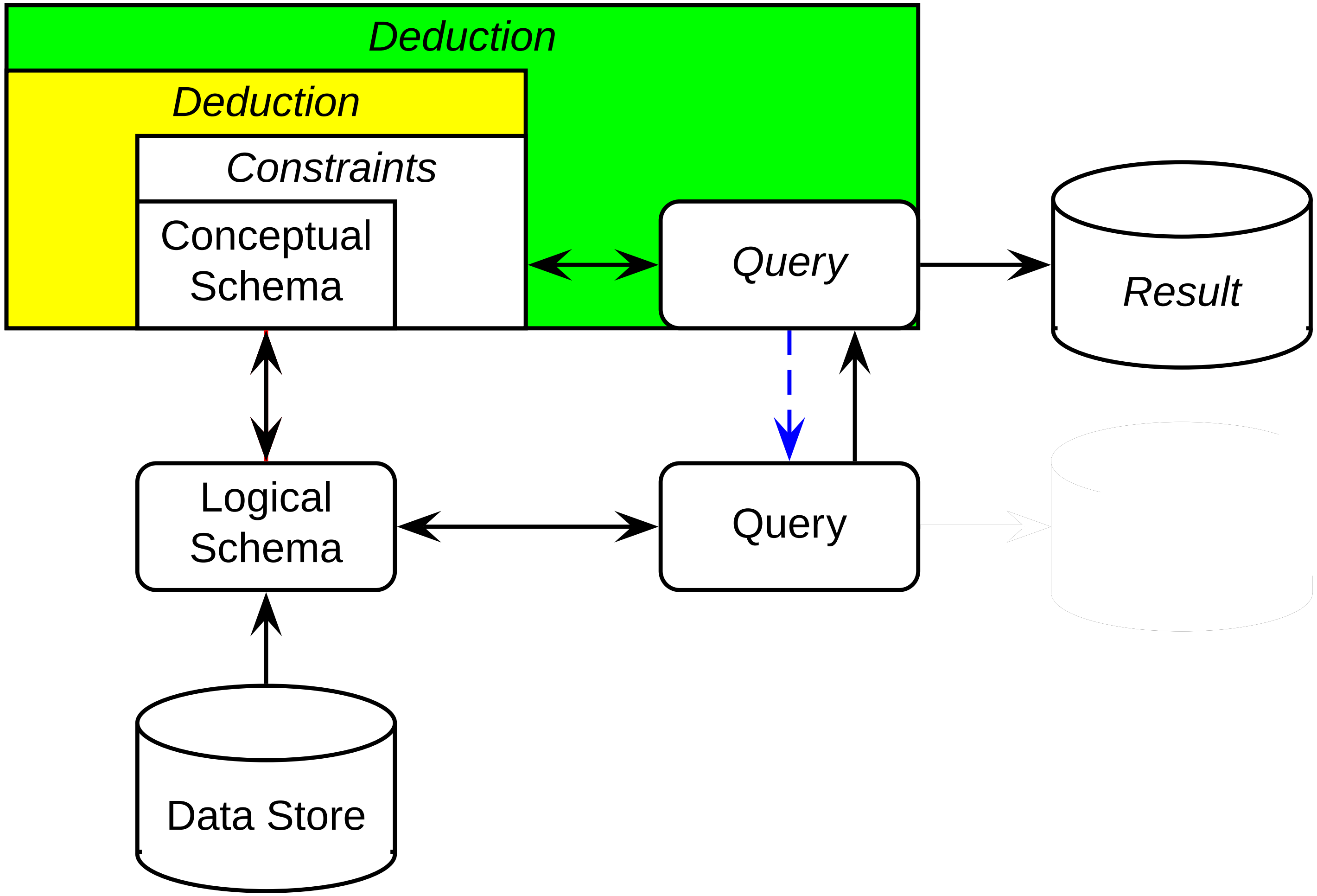
The role of a Conceptual Schema – revisited



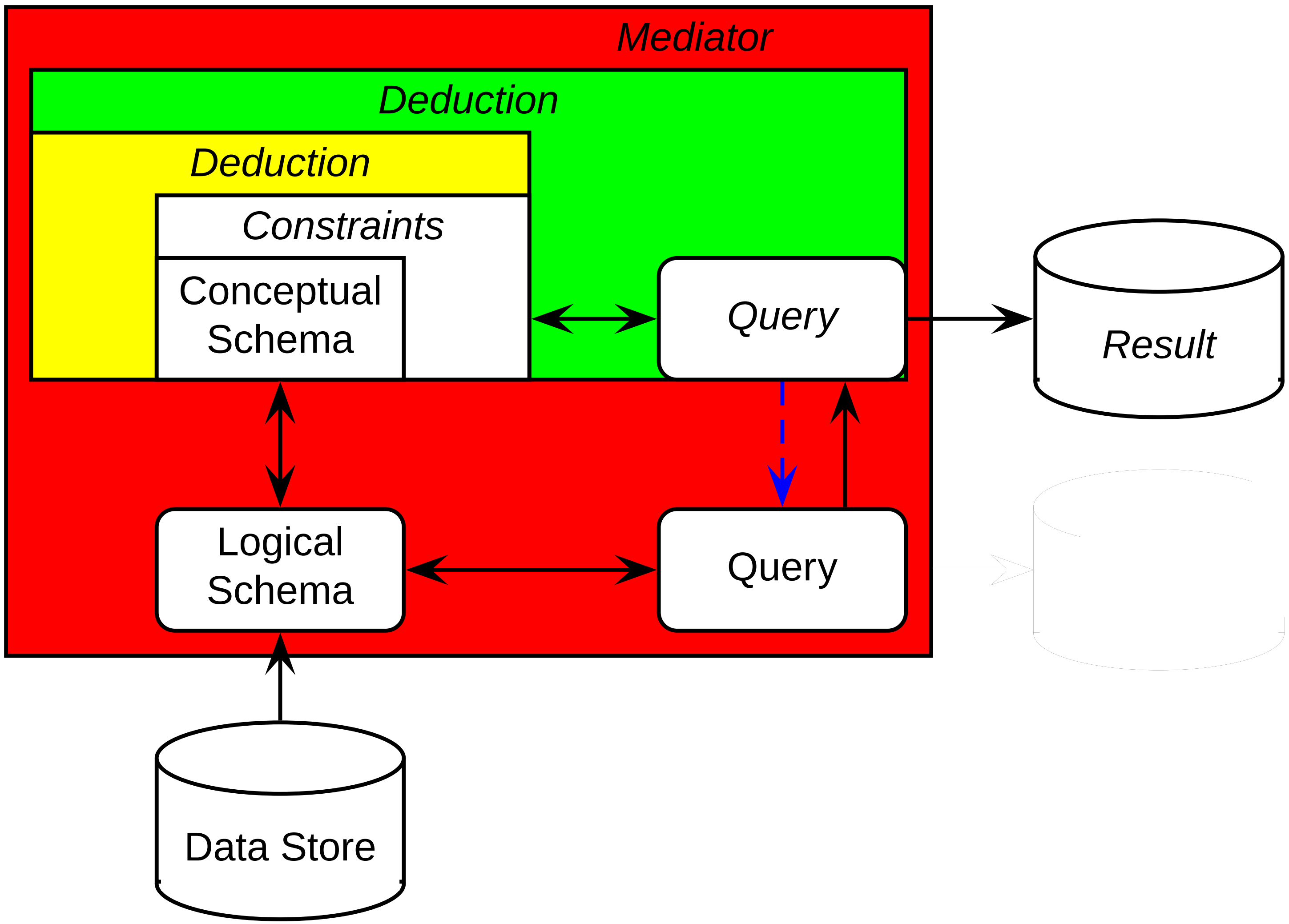
The role of a Conceptual Schema – revisited



The role of a Conceptual Schema – revisited



The role of a Conceptual Schema – revisited



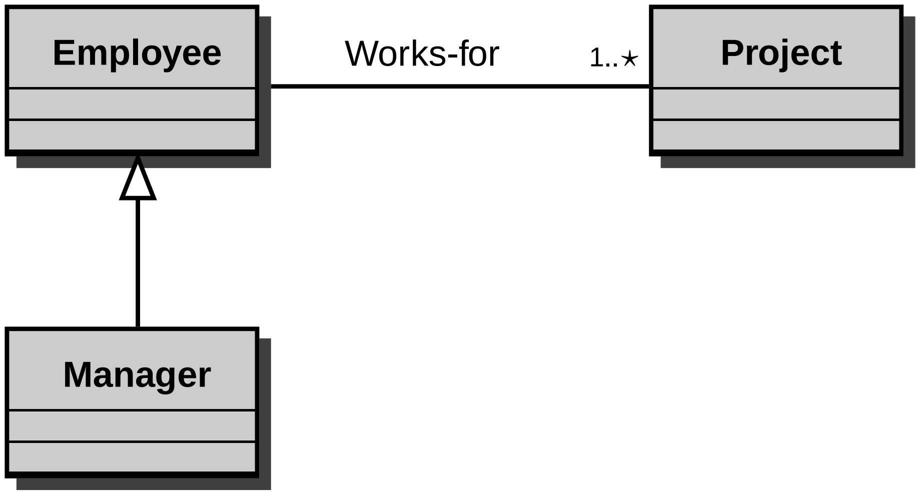
Queries with Ontologies: the DB assumption

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

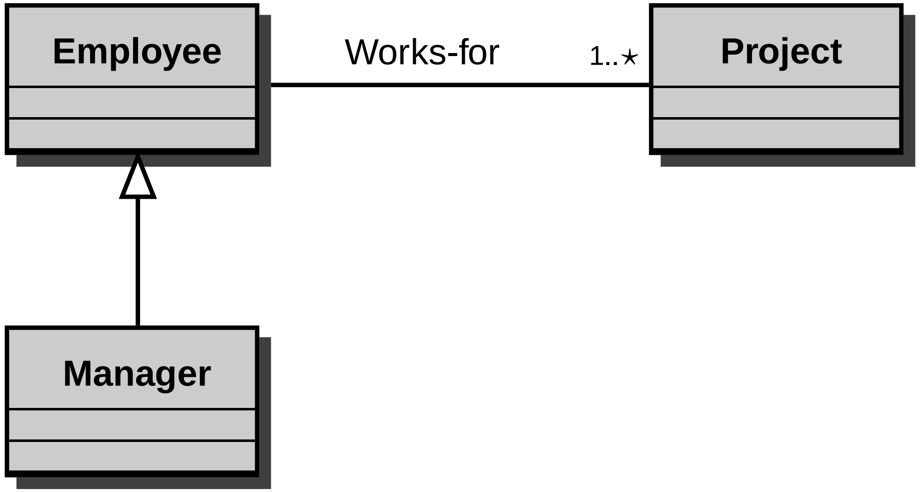
Queries with Ontologies: the DB assumption

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

Example with DB assumption

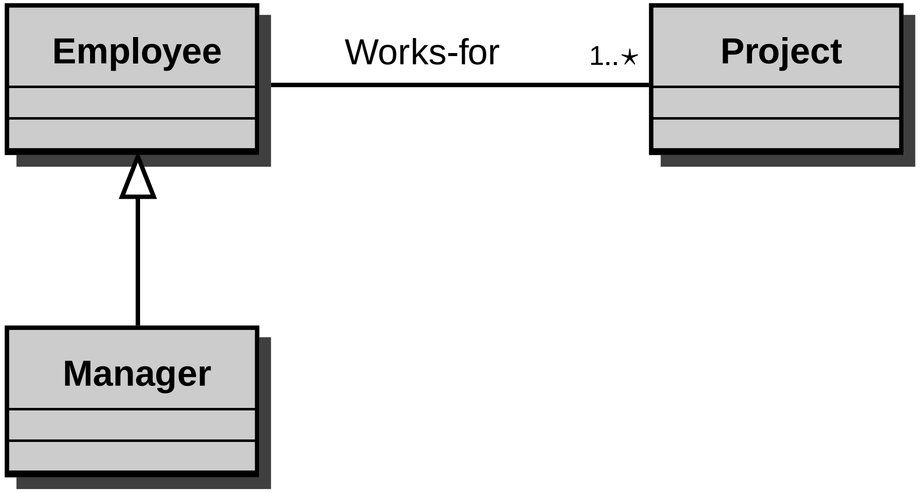


Example with DB assumption




```
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)

⟹ { John }
```

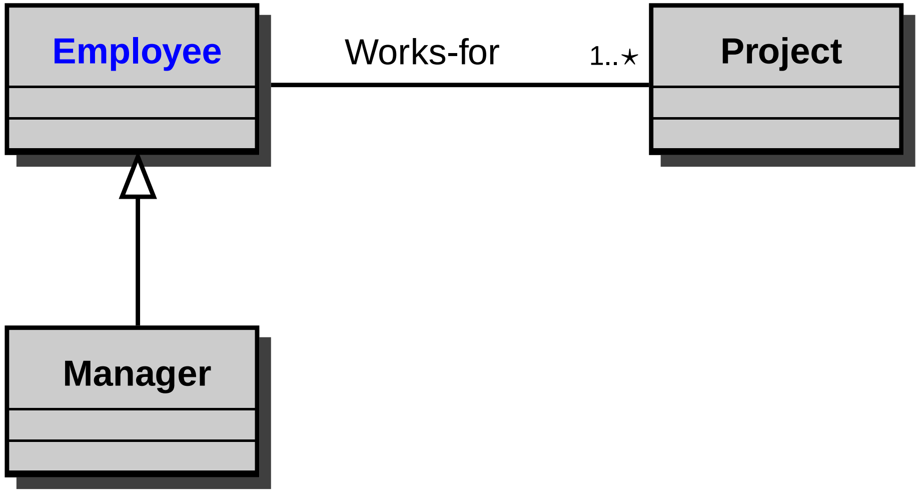
Weakening the DB assumption

- The DB assumption is against the principle that an ontology presents a richer vocabulary than the data stores.

Weakening the DB assumption

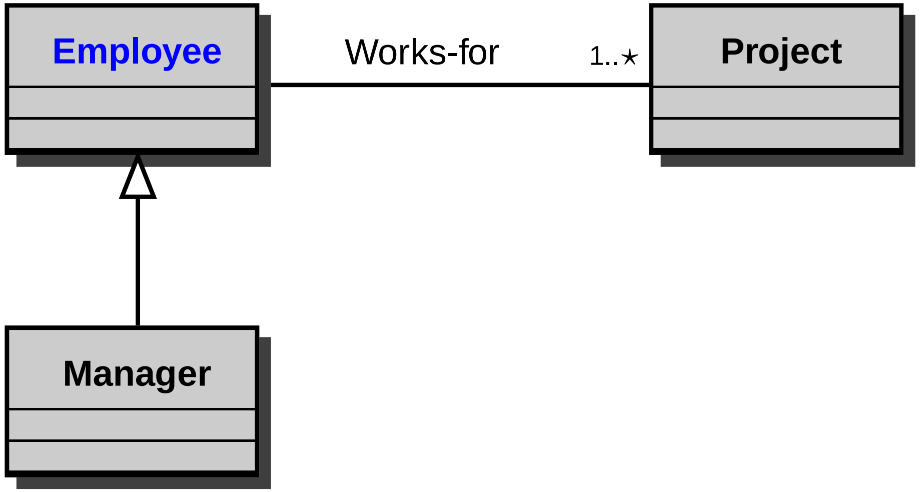
- The DB assumption is against the principle that an ontology presents a richer vocabulary than the data stores.
- Partial DB assumption: complete information **about *some* term** appearing in the ontology
- Standard DB technologies do not apply
- The query answering problem in this context is inherently 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



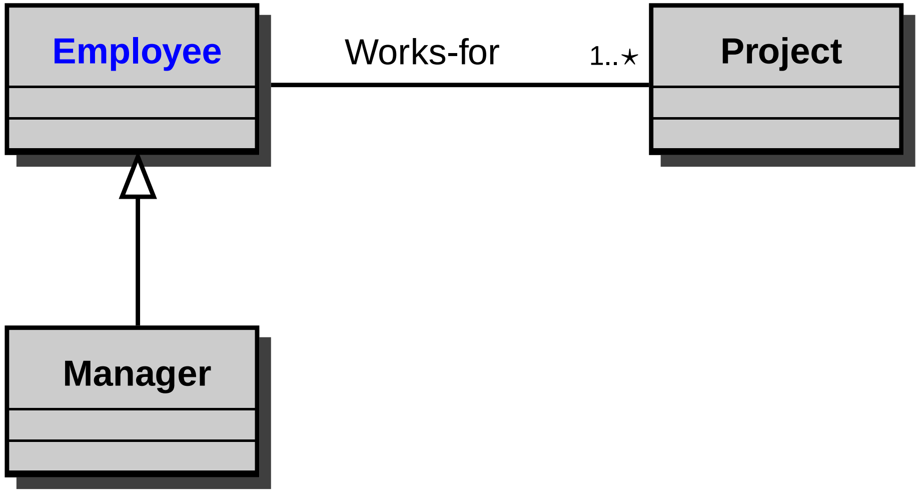
Manager = { **John**, **Paul** }

Works-for = { (**John**,**Prj-A**), (**Mary**,**Prj-B**) }

Project = { **Prj-A**, **Prj-B** }

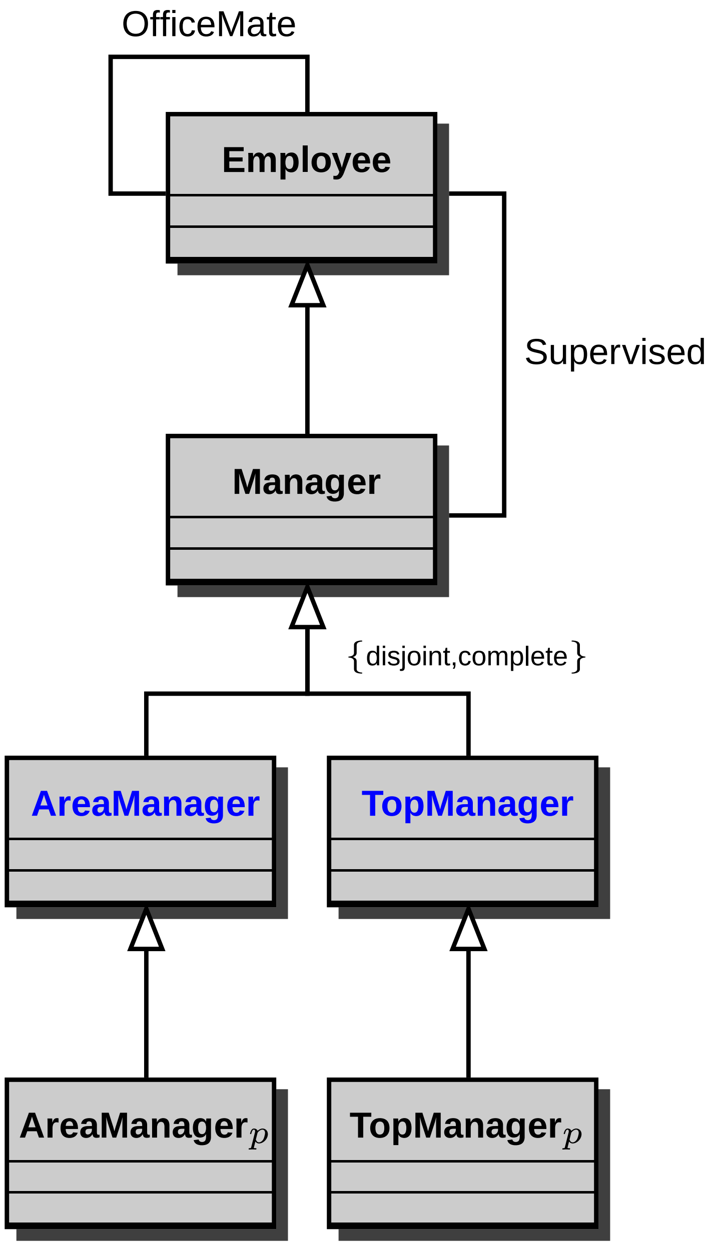
Q(X) :- Employee(X)

Example with partial DB assumption

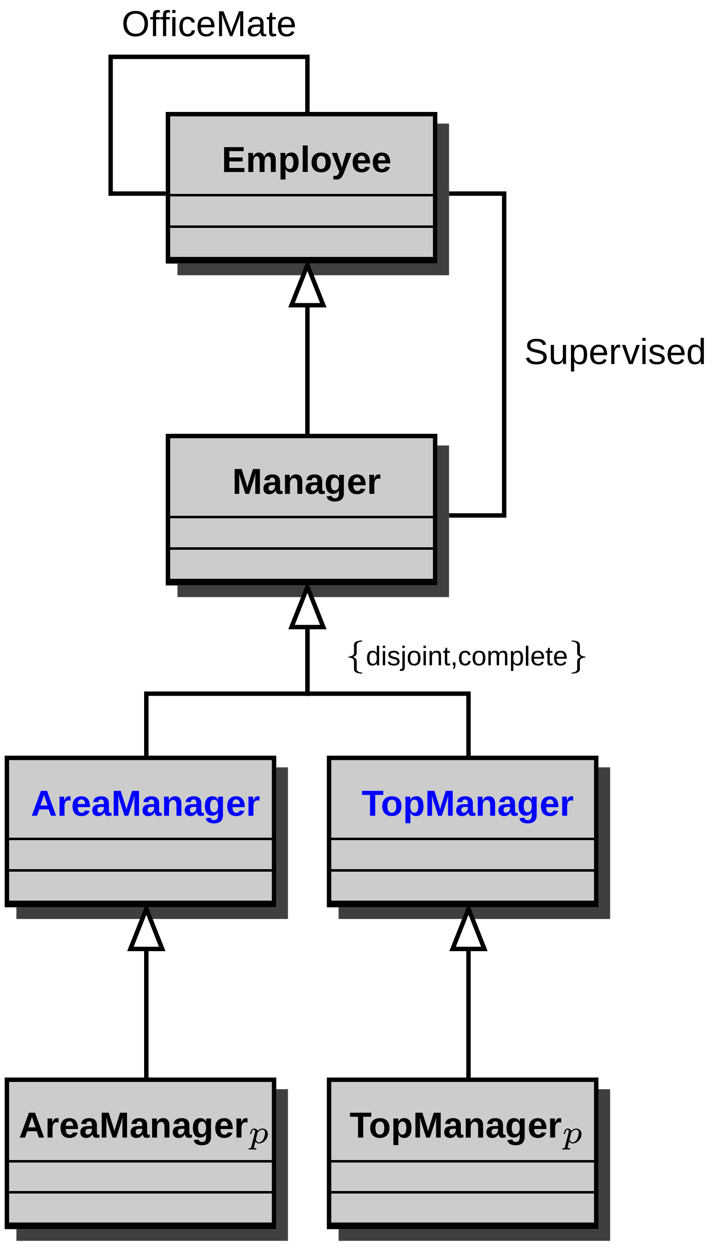



```
Manager = { John, Paul }

Works-for = { (John,Prj-A), (Mary,Prj-B) }

Project = { Prj-A, Prj-B }

Q(X) :- Employee(X)

⟹ { John, Paul, Mary }
```

Andrea’s Example

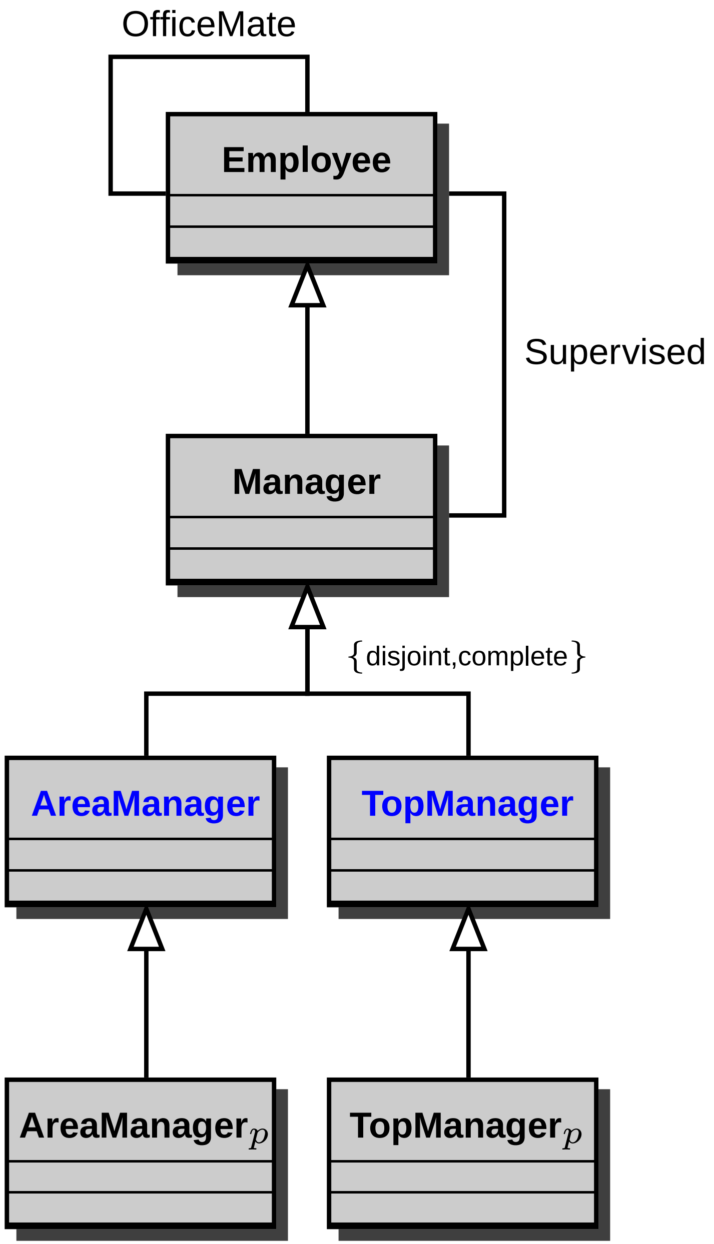


Andrea's Example

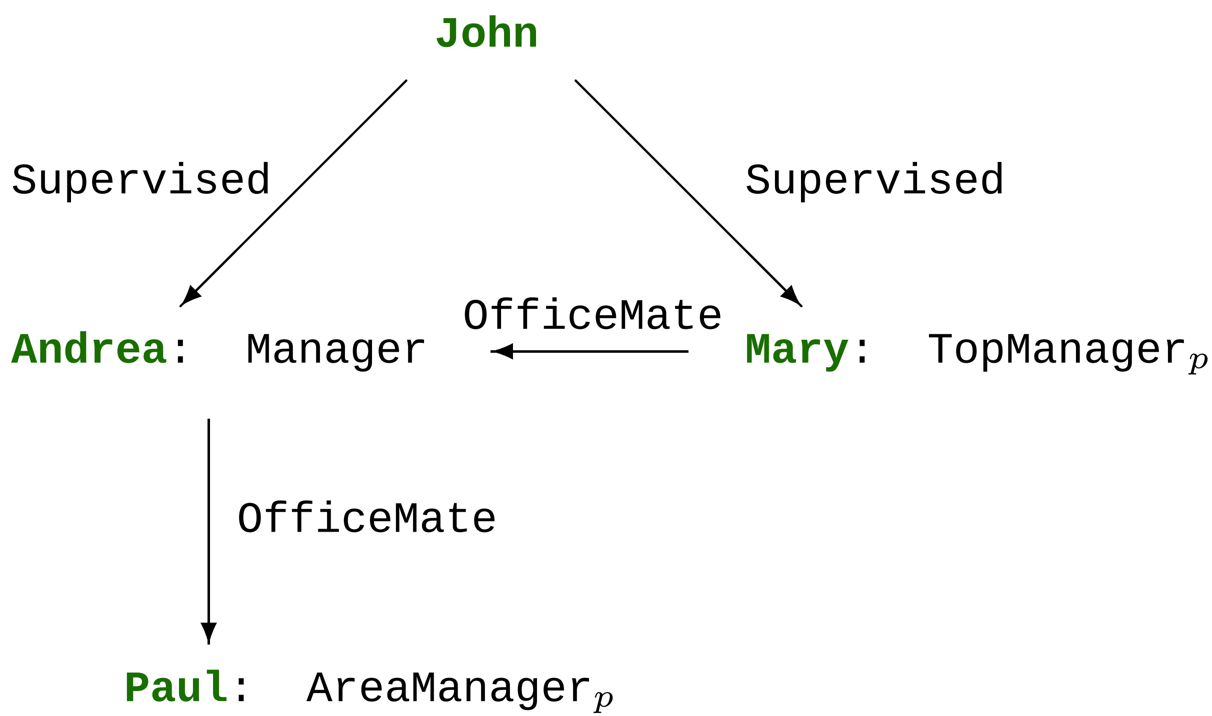


Employee = { **Andrea**, **Paul**, **Mary**, **John** }
Manager = { **Andrea**, **Paul**, **Mary**}
AreaManager$_p$ = { **Paul** }
TopManager$_p$ = { **Mary** }
Supervised = { (**John**,**Andrea**), (**John**,**Mary**) }
OfficeMate = { (**Mary**,**Andrea**), (**Andrea**,**Paul**) }

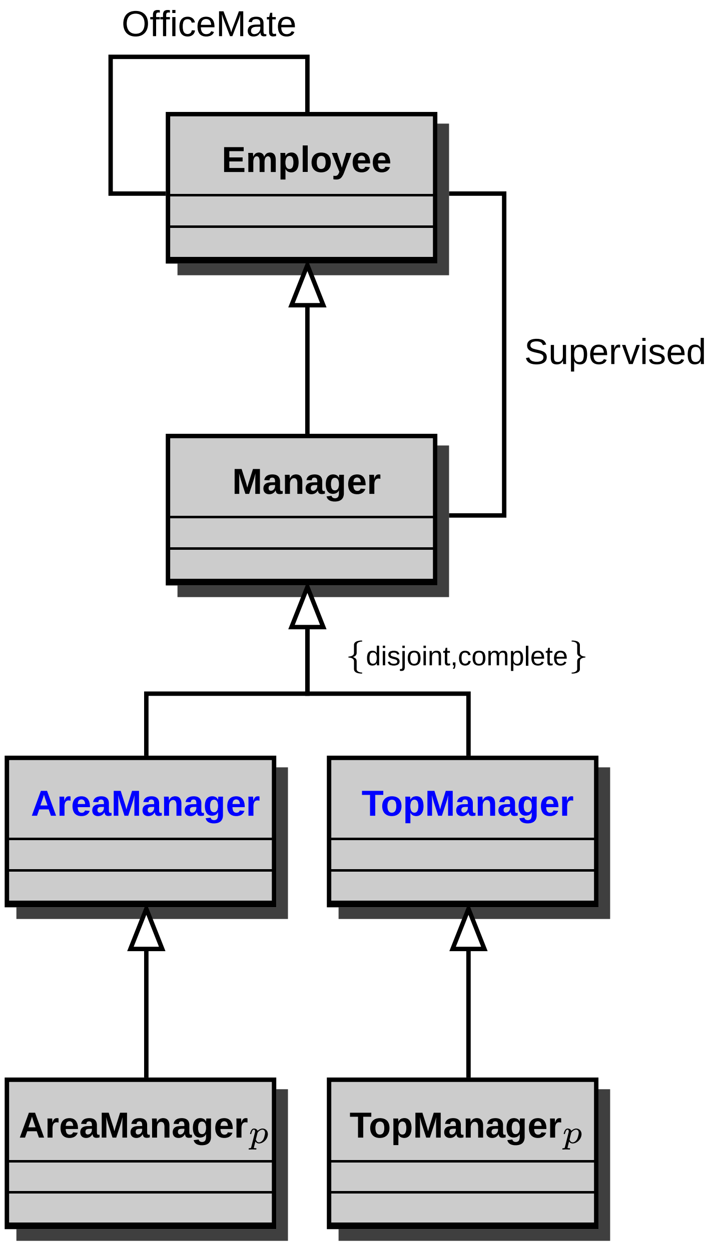
Andrea's Example



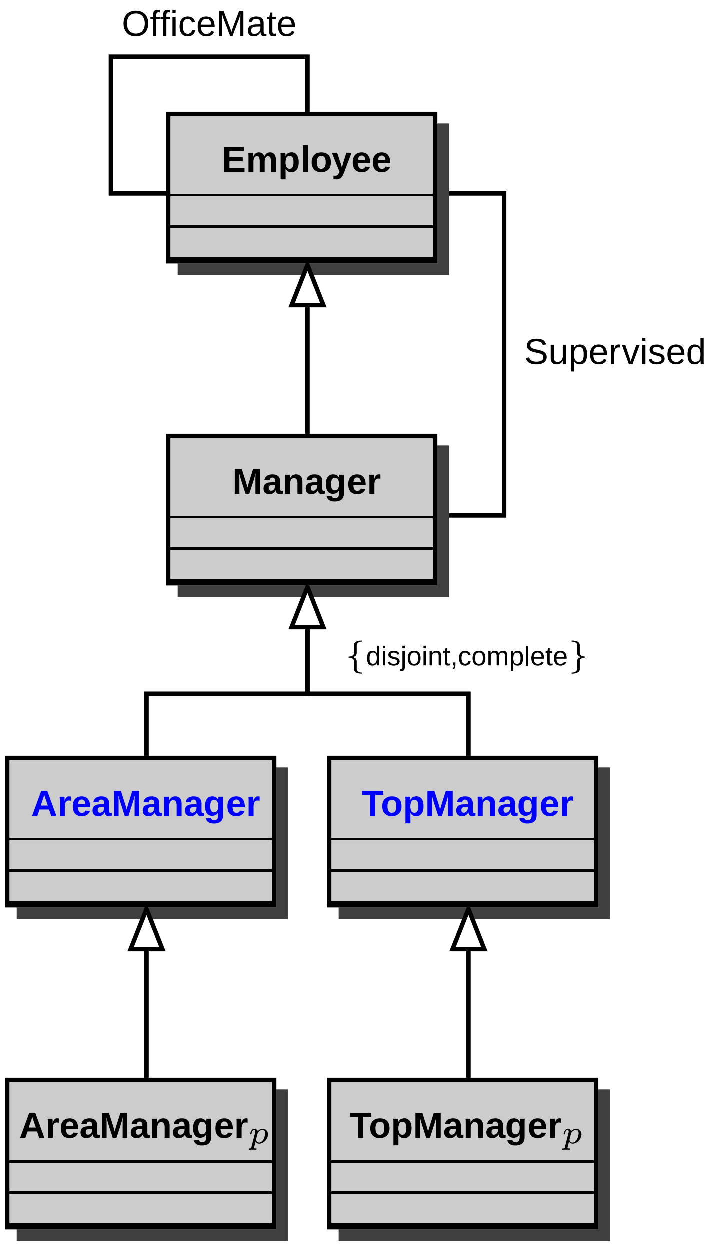
Employee = { **Andrea**, **Paul**, **Mary**, **John** }
Manager = { **Andrea**, **Paul**, **Mary**}
AreaManager$_p$ = { **Paul** }
TopManager$_p$ = { **Mary** }
Supervised = { (**John**,**Andrea**), (**John**,**Mary**) }
OfficeMate = { (**Mary**,**Andrea**), (**Andrea**,**Paul**) }

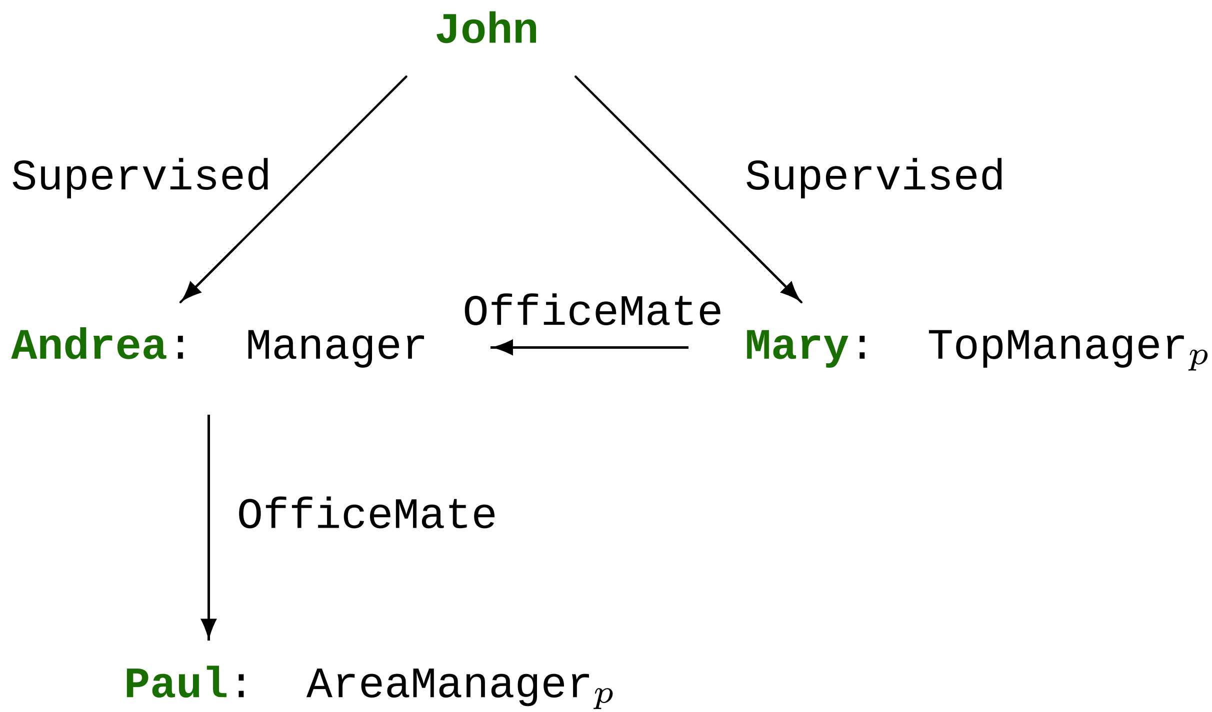


Andrea's Example (cont.)

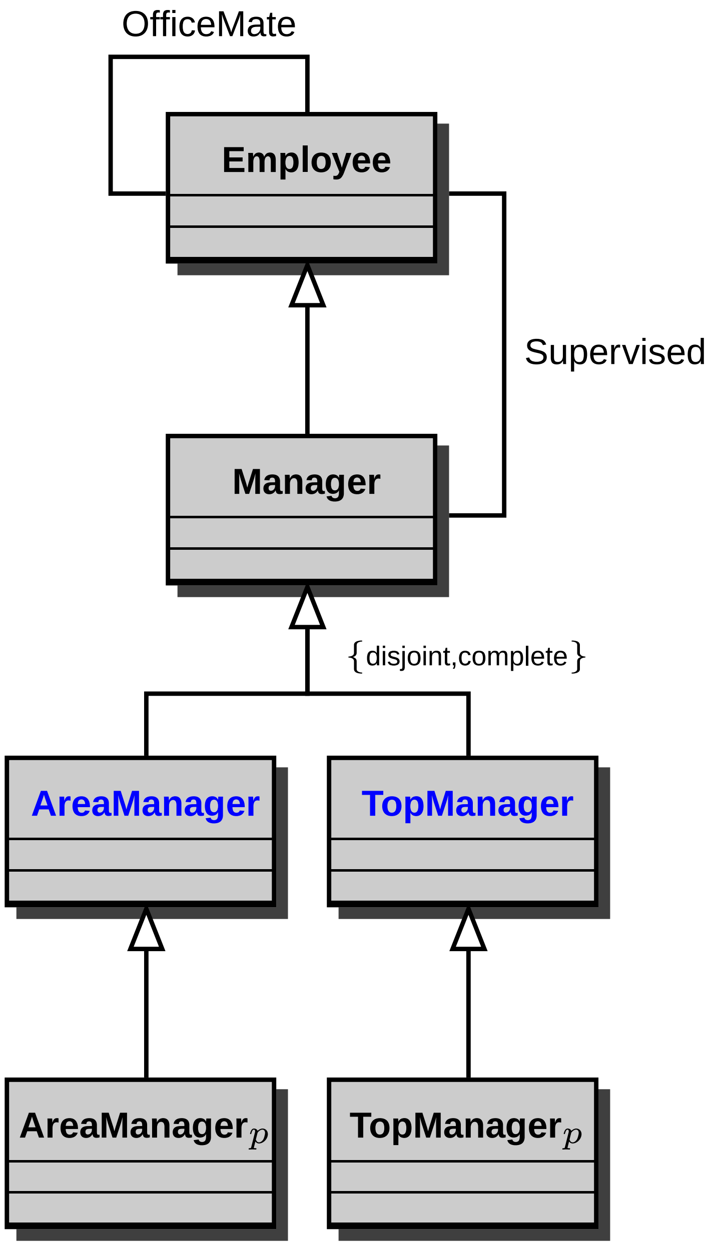


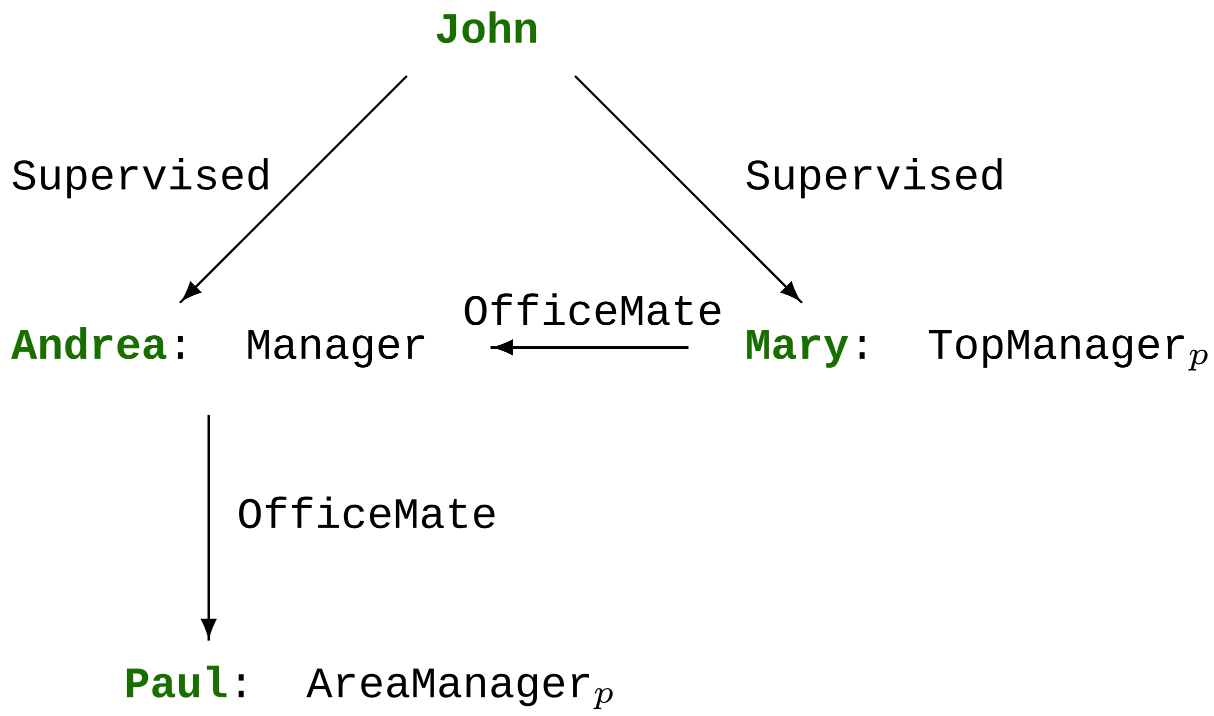
Andrea's Example (cont.)





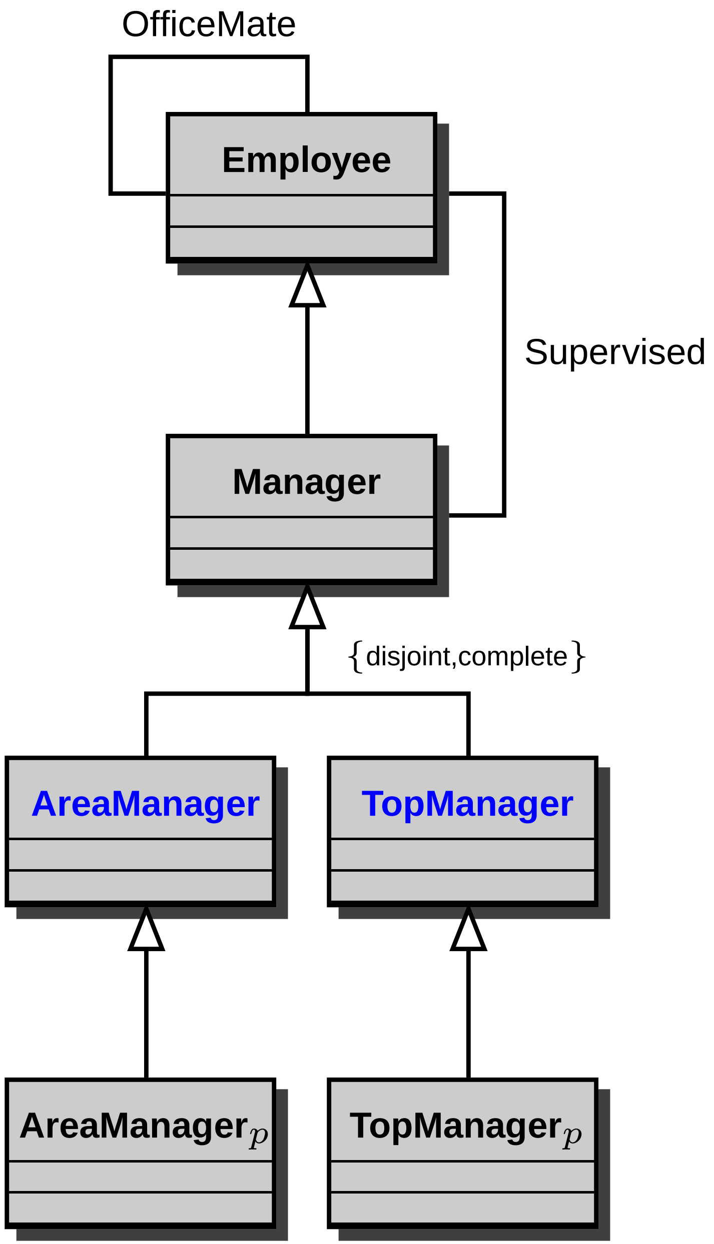
Andrea's Example (cont.)

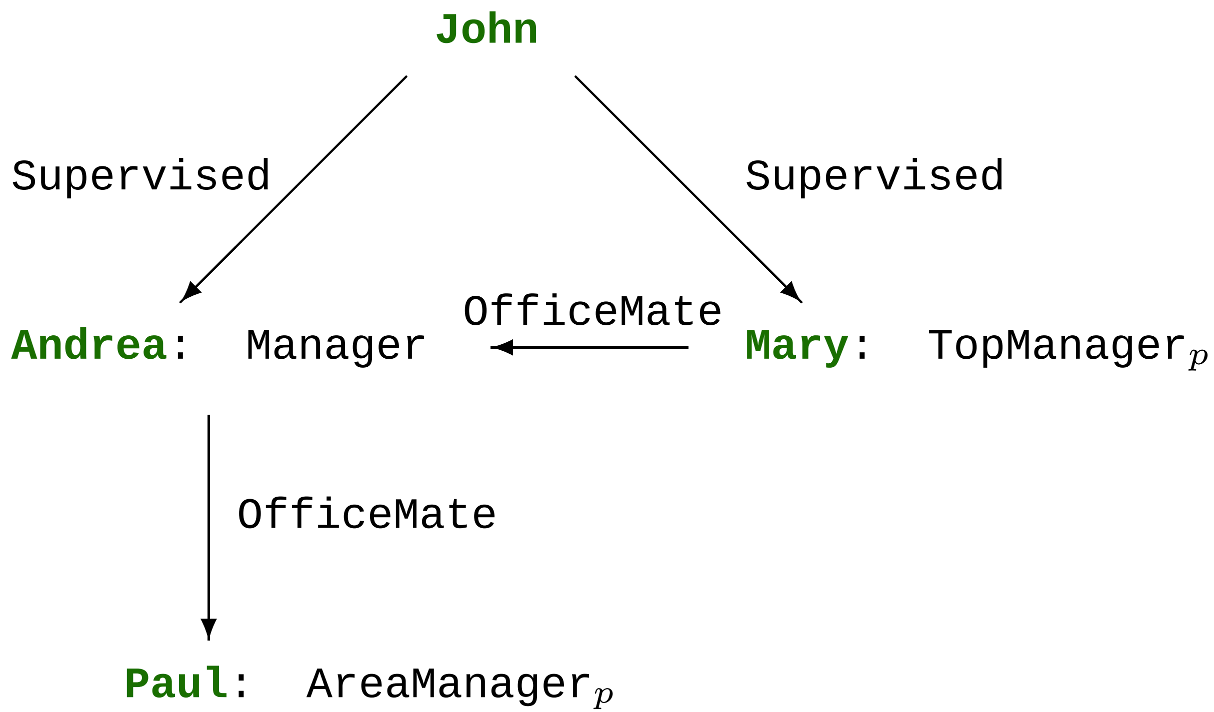




`Q(X) :- Supervised(X,Y), TopManager(Y),`
`Officemate(Y,Z), AreaManager(Z)`

Andrea's Example (cont.)






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

$\Longrightarrow$ { **John** }

The general case: View based Query Processing

In general, the **mapping** between the ontology and the information source terms can be given in terms of a set of sound or exact views:

The general case: View based Query Processing

In general, the **mapping** between the ontology and the information source terms can be given in terms of a set of sound or exact views:

- **GAV** (global-as-view): a view over the information source is given for some term in the ontology

The general case: View based Query Processing

In general, the **mapping** between the ontology and the information source terms can be given in terms of a set of sound or exact views:

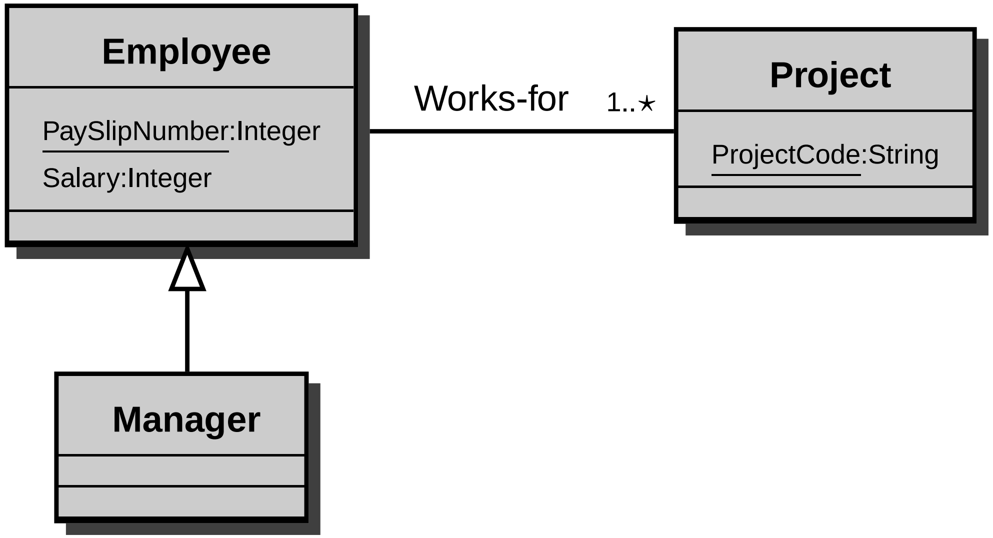
- **GAV** (global-as-view): a view over the information source is given for some term in the ontology
  - both the DB and the partial DB assumptions are special cases of GAV
  - an ER schema can be easily mapped to its corresponding relational schema in normal form via a GAV mapping

The general case: View based Query Processing

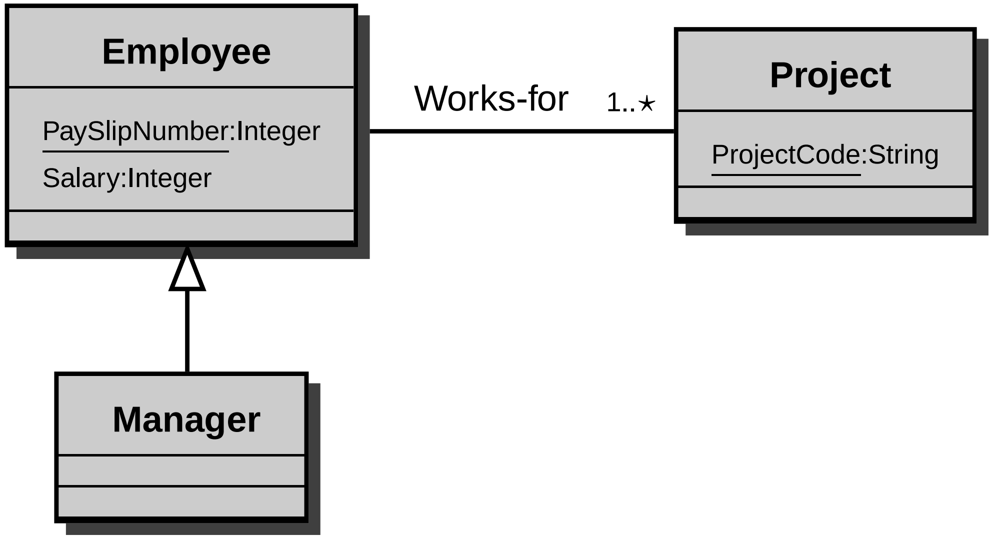
In general, the **mapping** between the ontology and the information source terms can be given in terms of a set of sound or exact views:

- **GAV** (global-as-view): a view over the information source is given for some term in the ontology
  - 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 normal form via a GAV mapping
- **LAV** (local-as-view): a view over the ontology terms is given for each term in the information source;
- **GLAV**: mixed from the above.

Sound GAV mapping



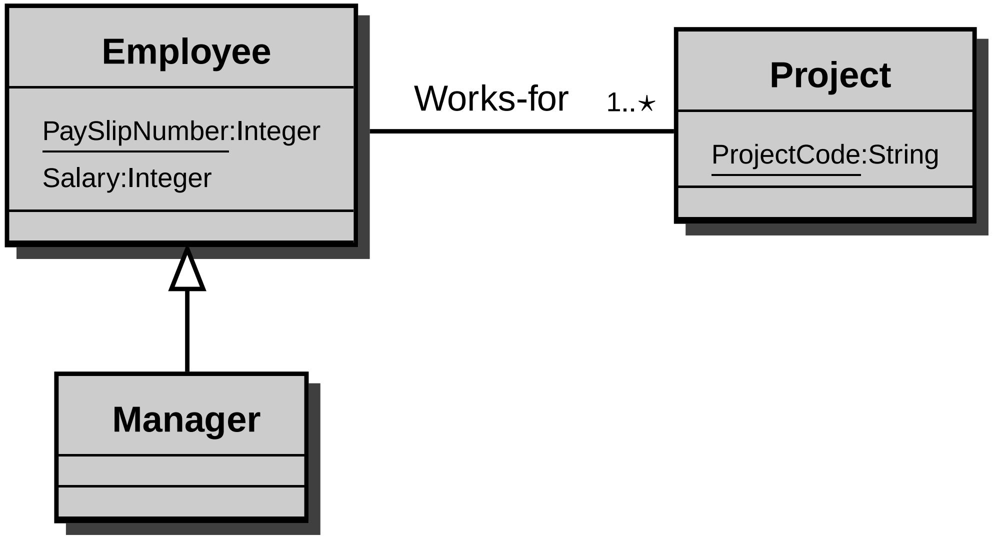
Sound GAV mapping



1-Employee(PaySlipNumber,Salary,ManagerP)

2-Works-for(PaySlipNumber,ProjectCode)

Sound GAV mapping



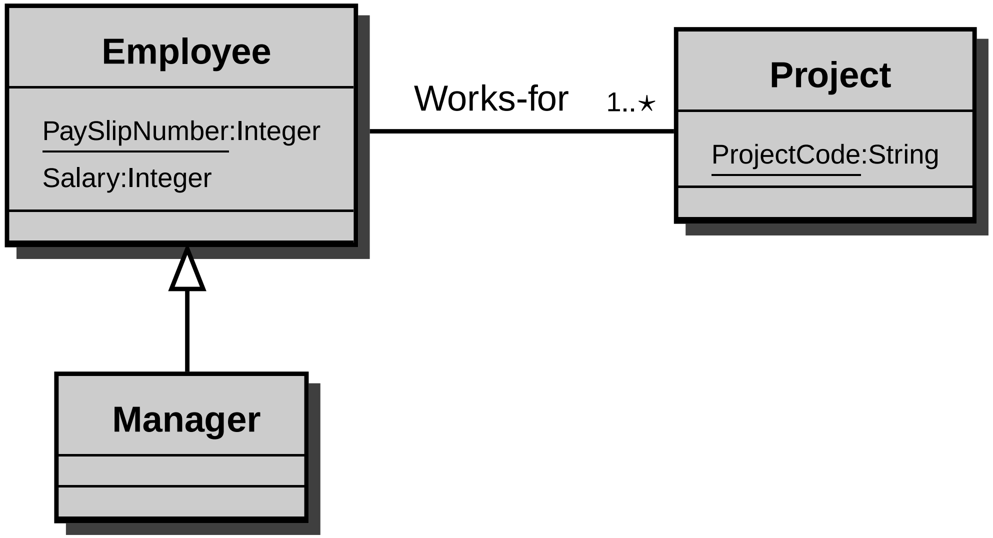
1-Employee(PaySlipNumber,Salary,ManagerP)

2-Works-for(PaySlipNumber,ProjectCode)

```
Employee(X) :- 1-Employee(X,Y,Z)
Employee(X) :- 2-Works-for(X,Y)
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)
```

Queries with Sound GAV mapping

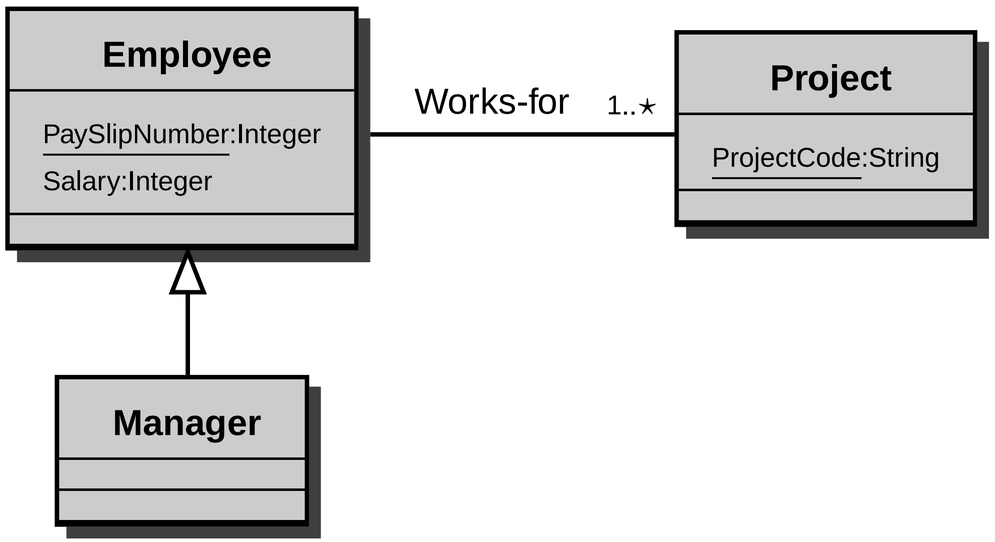


1-Employee(PaySlipNumber,Salary,ManagerP)

2-Works-for(PaySlipNumber,ProjectCode)

```
Employee(X) :- 1-Employee(X,Y,Z)
Employee(X) :- 2-Works-for(X,Y)
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)
```

Queries with Sound GAV mapping



1-Employee(PaySlipNumber,Salary,ManagerP)

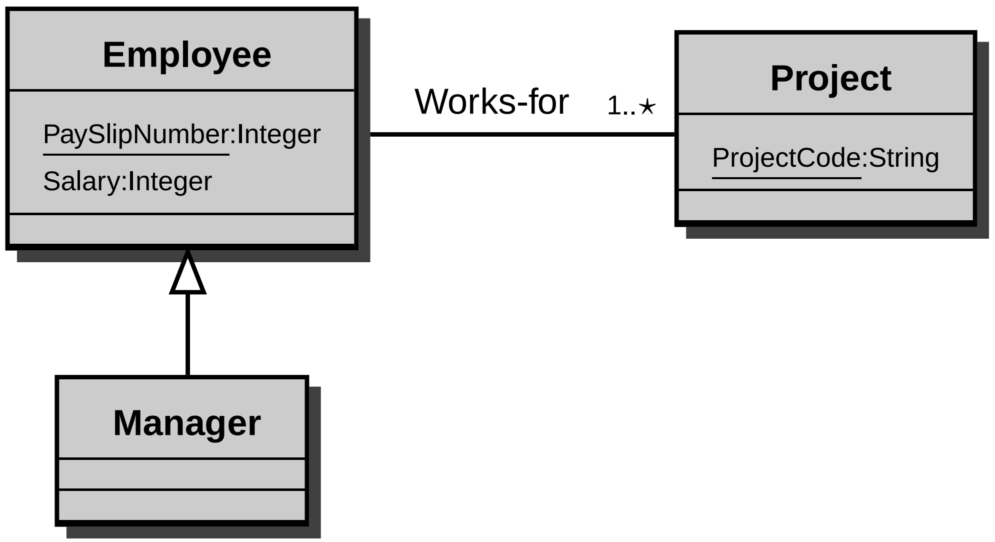
2-Works-for(PaySlipNumber,ProjectCode)

Employee(X) :- 1-Employee(X,Y,Z)
Employee(X) :- 2-Works-for(X,Y)
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)

Queries with Sound GAV mapping



1-Employee(PaySlipNumber,Salary,ManagerP)

2-Works-for(PaySlipNumber,ProjectCode)

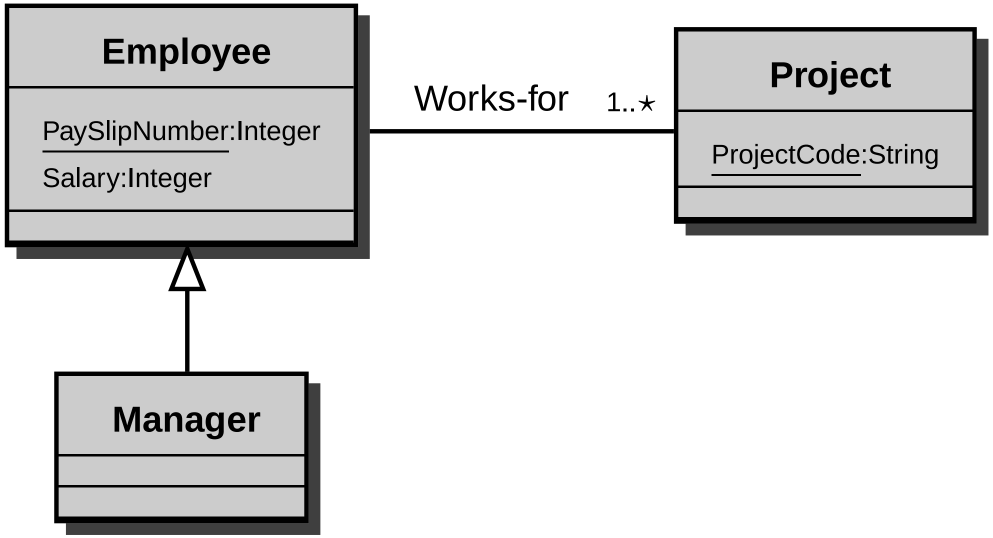
Employee(X) :- 1-Employee(X,Y,Z)
Employee(X) :- 2-Works-for(X,Y)
Manager(X) :- 1-Employee(X,Y, true)

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

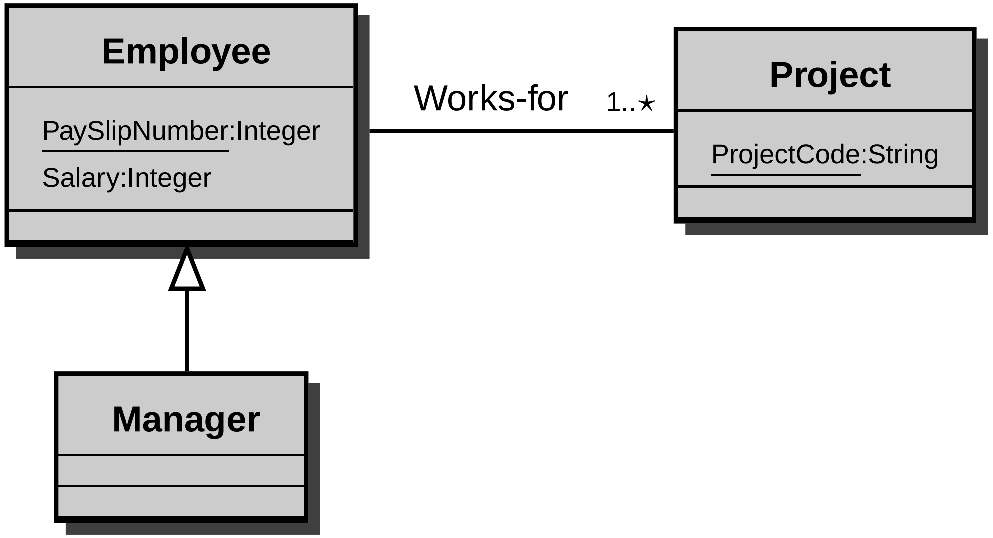
Q(X) :- Employee(X)

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

Sound LAV mapping



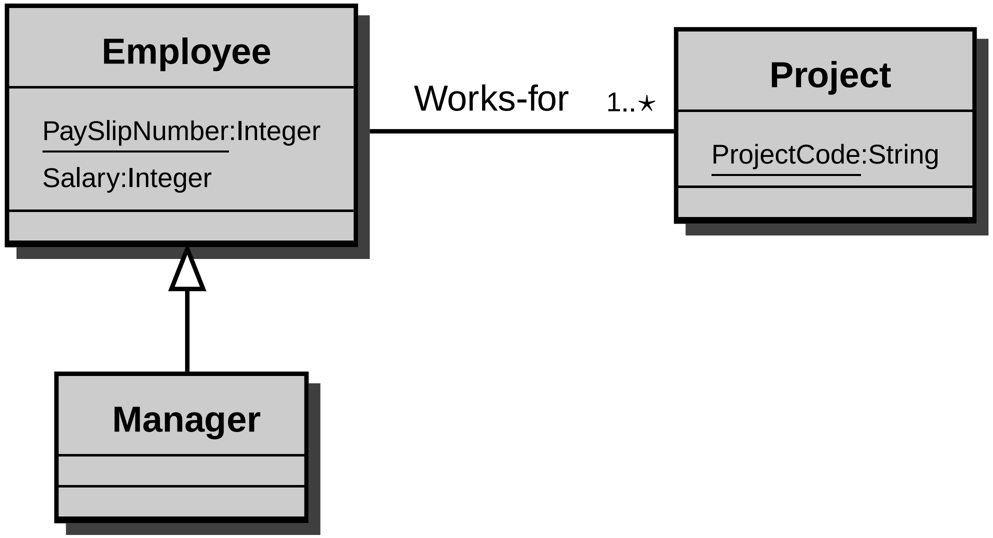
Sound LAV mapping



1-Employee(<u>PaySlipNumber</u>,Salary,ManagerP)

2-Works-for(<u>PaySlipNumber</u>,<u>ProjectCode</u>)

Sound LAV mapping



1-Employee(<u>PaySlipNumber</u>,Salary,ManagerP)

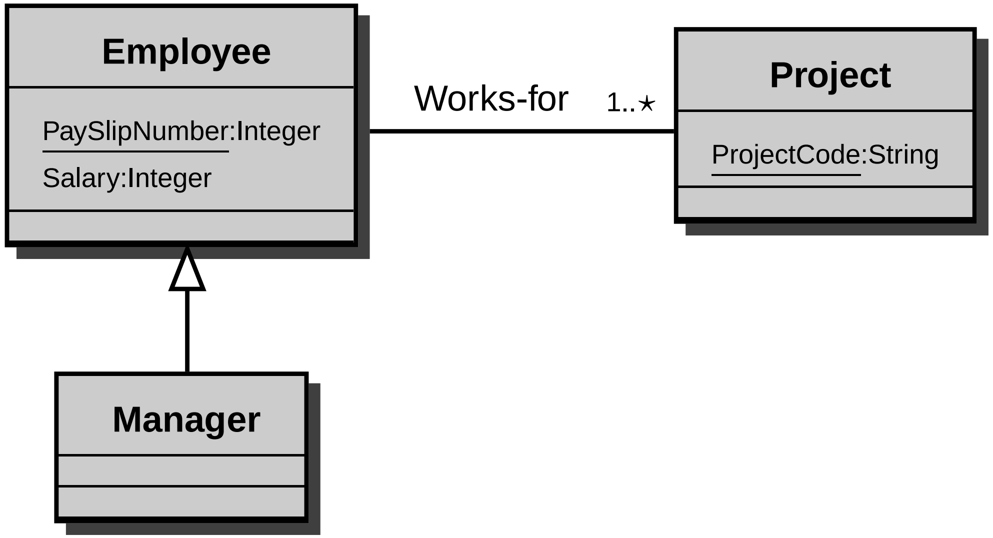
2-Works-for(<u>PaySlipNumber</u>,<u>ProjectCode</u>)

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)

Queries with Sound LAV mapping



1-Employee(<u>PaySlipNumber</u>,Salary,ManagerP)

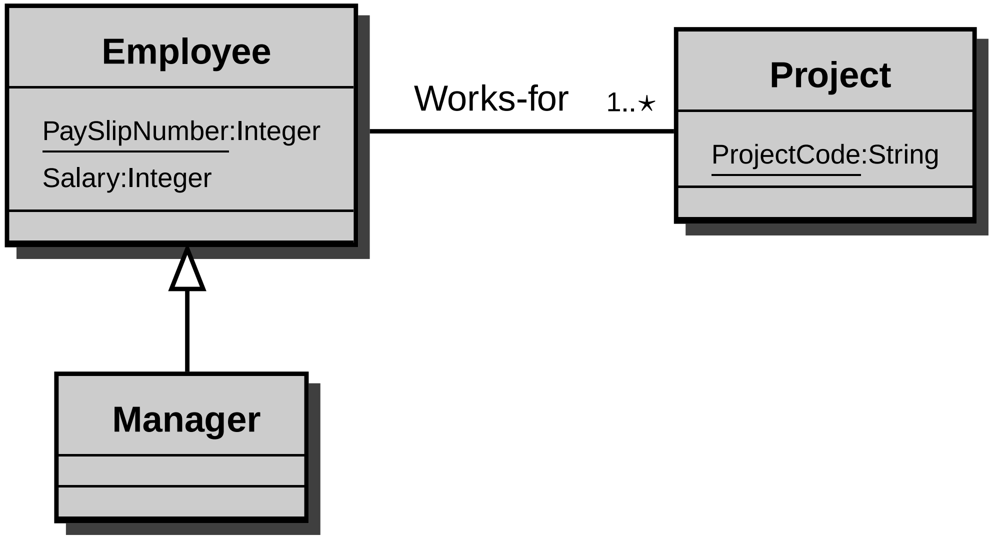
2-Works-for(<u>PaySlipNumber</u>,<u>ProjectCode</u>)

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)

Queries with Sound LAV mapping



1-Employee(<u>PaySlipNumber</u>,Salary,ManagerP)

2-Works-for(<u>PaySlipNumber</u>,<u>ProjectCode</u>)

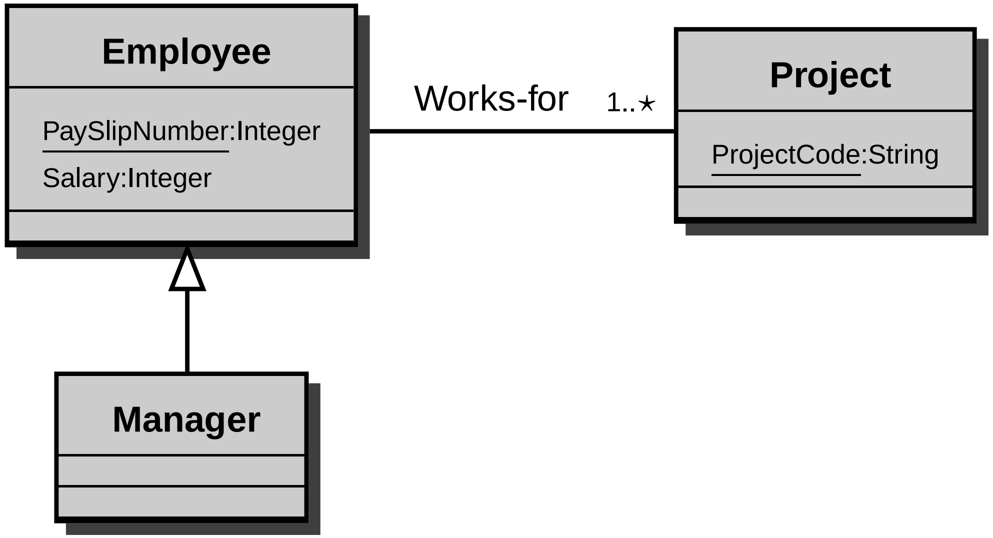
1-Employee(X,Y,Z) :- Manager(X), Salary(X,Y), Z=true

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)

Queries with 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

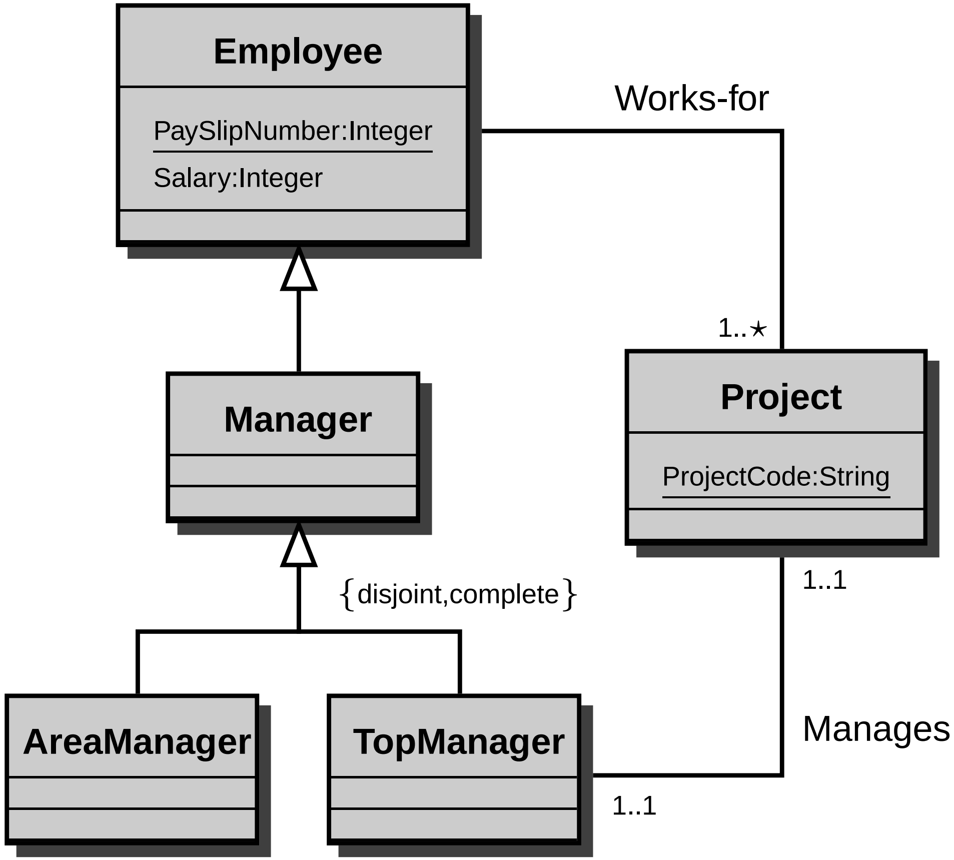
2-Works-for(X,Y) :- Works-for(X,Y)

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

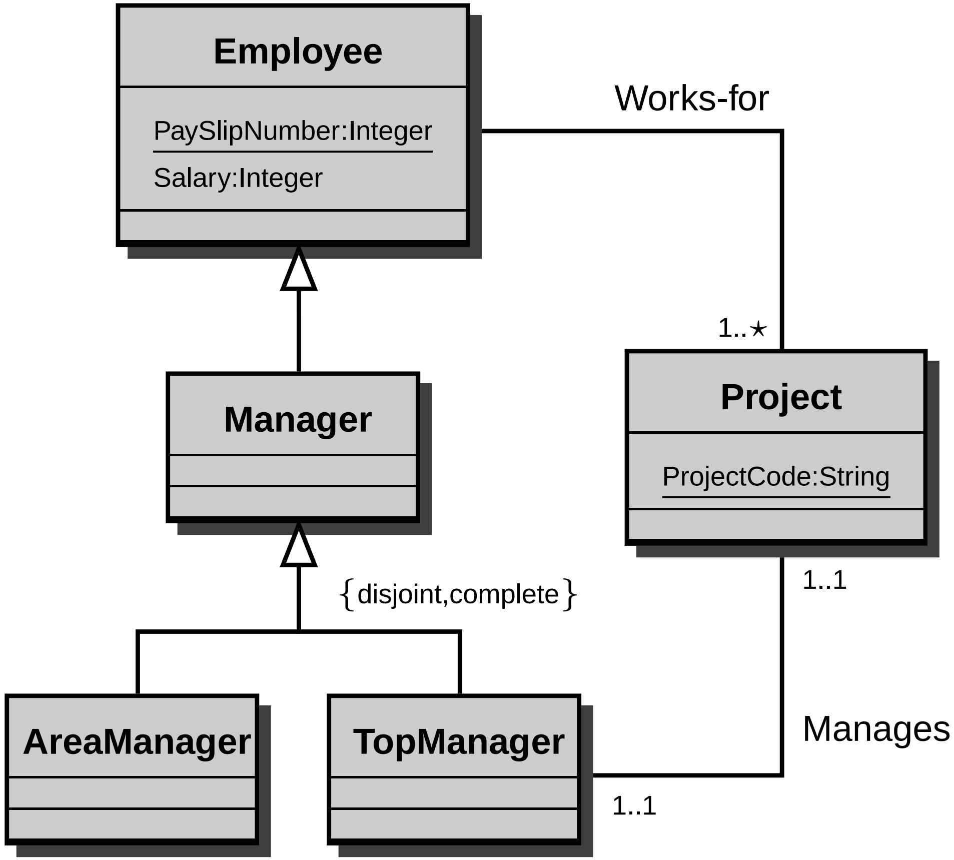
```
Q(X,Y) :- Employee(X), Works-for(X,Y), Manages(X,Y)
```



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

Reasoning over queries

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



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

⇝ INCONSISTENT QUERY!

Local-as-view vs. Global-as-view

Local-as-view

- High modularity and reusability (when a source changes, only its view definition is changed).
- Relationships between sources can be inferred.
- Computationally more difficult (query reformulation).

Global-as-view

- Whenever the source changes or a new one is added, the view needs to be reconsidered.
- Needs to understand the relationships between the sources.
- Query processing sometimes easy (unfolding), when the ontology is *very simple*. Otherwise it requires sophisticated query evaluation procedures.

Possible scenarios

- Empty ontology / very simple Ontology
  - Global-as-view
  - Local-as-view
- Full Ontology / Integrity Constraints
  - Global-as-view
  - Local-as-view

Possible scenarios

- Empty ontology / very simple Ontology
  - Global-as-view
    - The problem reduces to standard DB technology.
    - Can not express Ontology Integration needs.
    - Not modular.
  - Local-as-view
- Full Ontology / Integrity Constraints
  - Global-as-view
  - Local-as-view

Possible scenarios

- Empty ontology / very simple Ontology
  - Global-as-view
    - The problem reduces to standard DB technology.
    - Can not express Ontology Integration needs.
    - Not modular.
  - Local-as-view
    - “Standard” view-based query processing.
    - Can express only few Ontology Integration needs.
    - Modular.
- Full Ontology / Integrity Constraints
  - Global-as-view
  - Local-as-view

Possible scenarios

- Empty ontology / very simple Ontology
  - Global-as-view
    - The problem reduces to standard DB technology.
    - Can not express Ontology Integration needs.
    - Not modular.
  - Local-as-view
    - “Standard” view-based query processing.
    - Can express only few Ontology Integration needs.
    - Modular.
- Full Ontology / Integrity Constraints
  - Global-as-view
    - Requires sophisticated query evaluation procedures (involving deduction).
    - Can express Ontology Integration needs.
    - Not modular.
  - Local-as-view

Possible scenarios

- Empty ontology / very simple Ontology
  - Global-as-view
    - The problem reduces to standard DB technology.
    - Can not express Ontology Integration needs.
    - Not modular.
  - Local-as-view
    - “Standard” view-based query processing.
    - Can express only few Ontology Integration needs.
    - Modular.
- Full Ontology / Integrity Constraints
  - Global-as-view
    - Requires sophisticated query evaluation procedures (involving deduction).
    - Can express Ontology Integration needs.
    - Not modular.
  - Local-as-view
    - View-based query processing under constraints.
    - Can express Ontology Integration needs.
    - Modular.

Current Practice

- Most implemented ontology based systems:

Current Practice

- Most implemented ontology based systems:
  - either assume no Ontology or a very simple Ontology with a global-as-view approach,

Current Practice

- Most implemented ontology based systems:
  - either assume no Ontology or a very simple Ontology with a global-as-view approach,
  - or include an Ontology or Integrity Constraints in their framework, but adopt a naive query evaluation procedure, based on query unfolding: no correctness of the query answering can be proved.

Conclusions

Conclusions

Do you have an ontology in your application?

Conclusions

Do you have an ontology in your application?

Pay attention!