Description Logics for Conceptual Design, Information Access, and Ontology Integration

Tutorial, 2002

Enrico Franconi

franconi@inf.unibz.it

http://www.inf.unibz.it/˜franconi

Faculty of Computer Science, Free University of Bozen-Bolzano

Summary

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

Summary

- Logic and Conceptual Modelling
- 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.

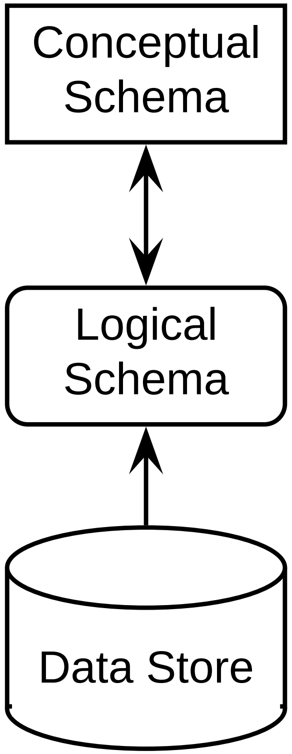
What is an Ontology

- 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.

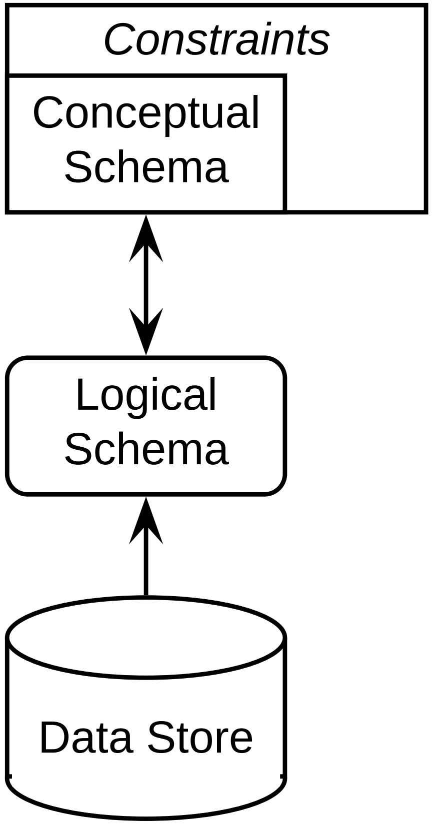
What is an Ontology

- 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.

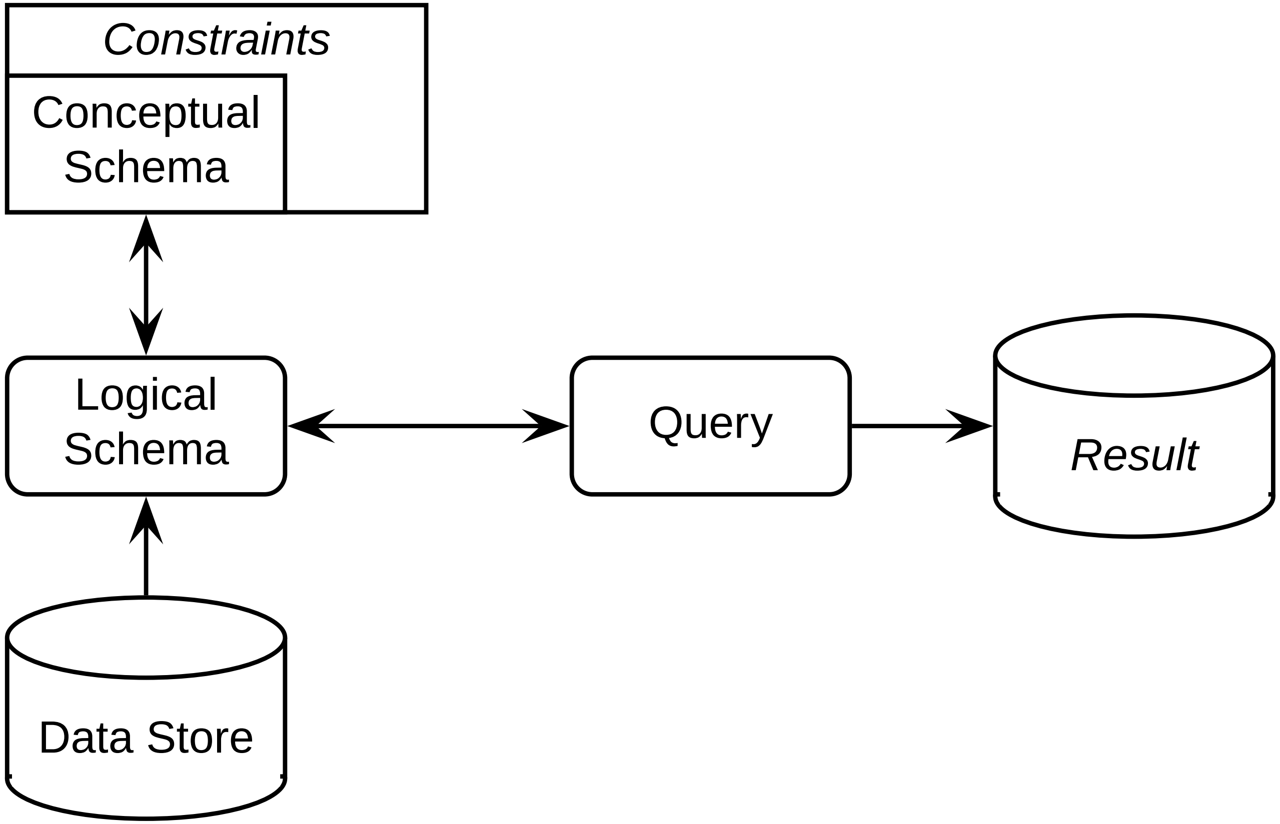
The role of a Conceptual Schema



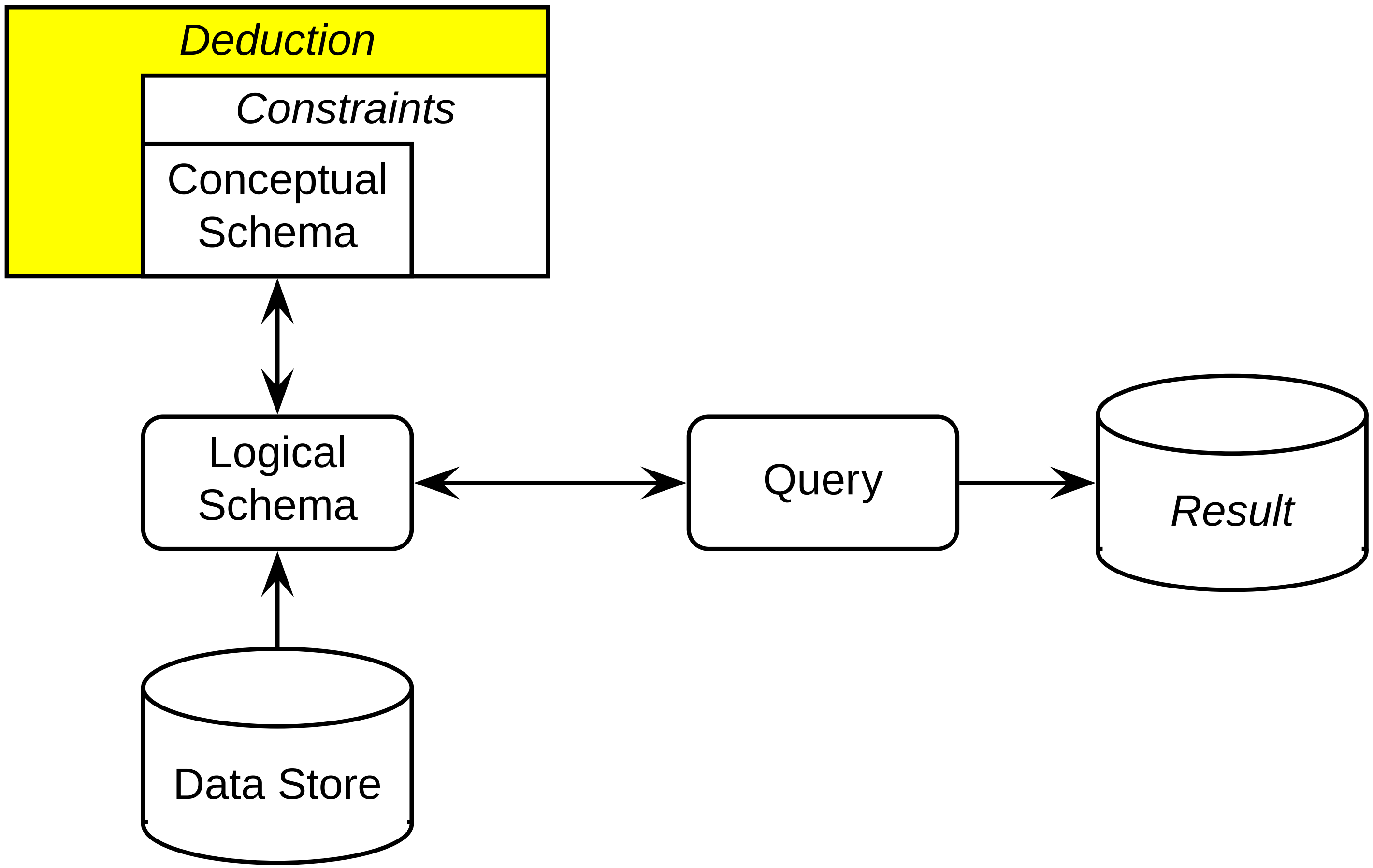
The role of a Conceptual Schema



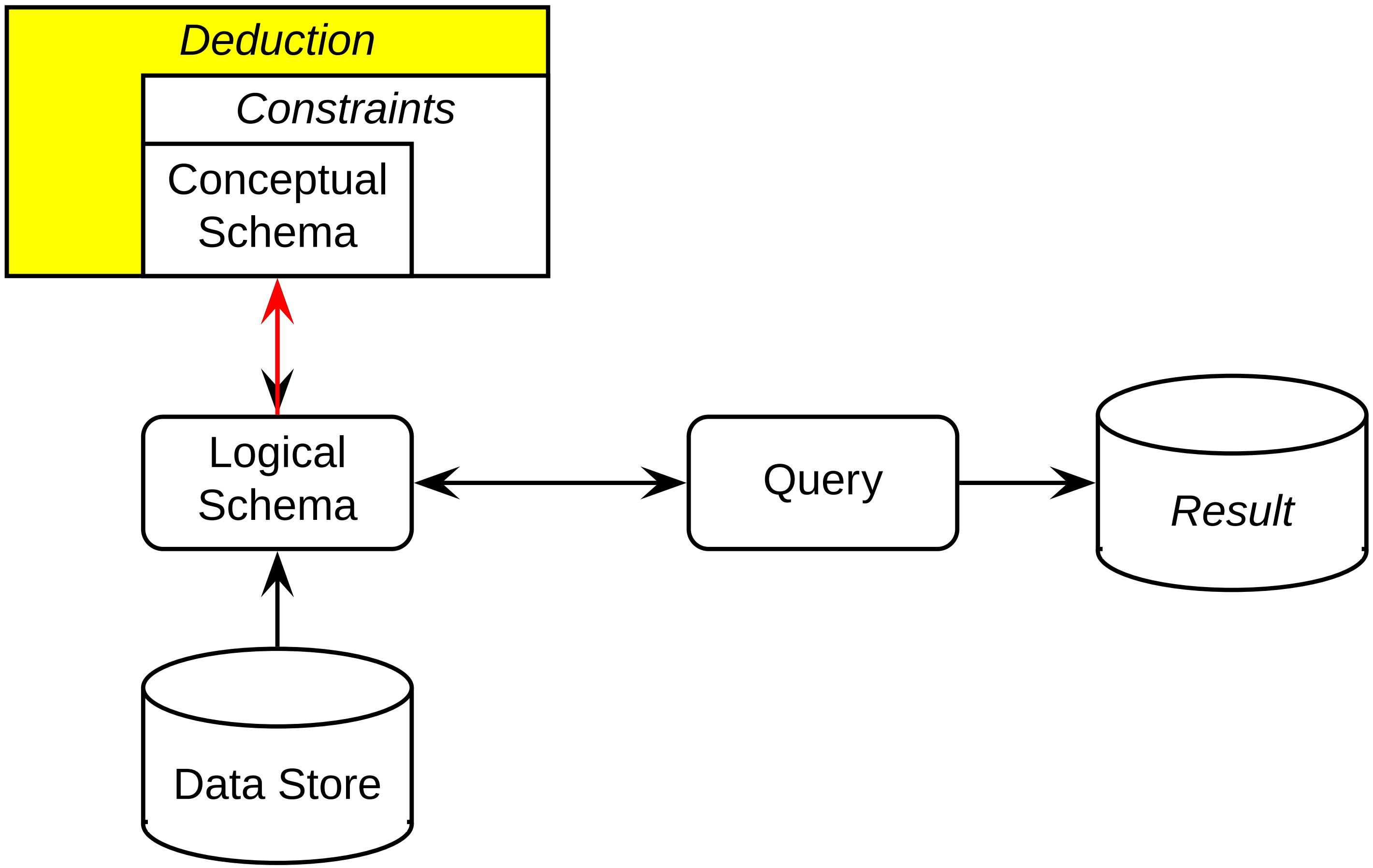
The role of a Conceptual Schema



The role of a Conceptual Schema



The role of a Conceptual Schema



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 and DAML+OIL).

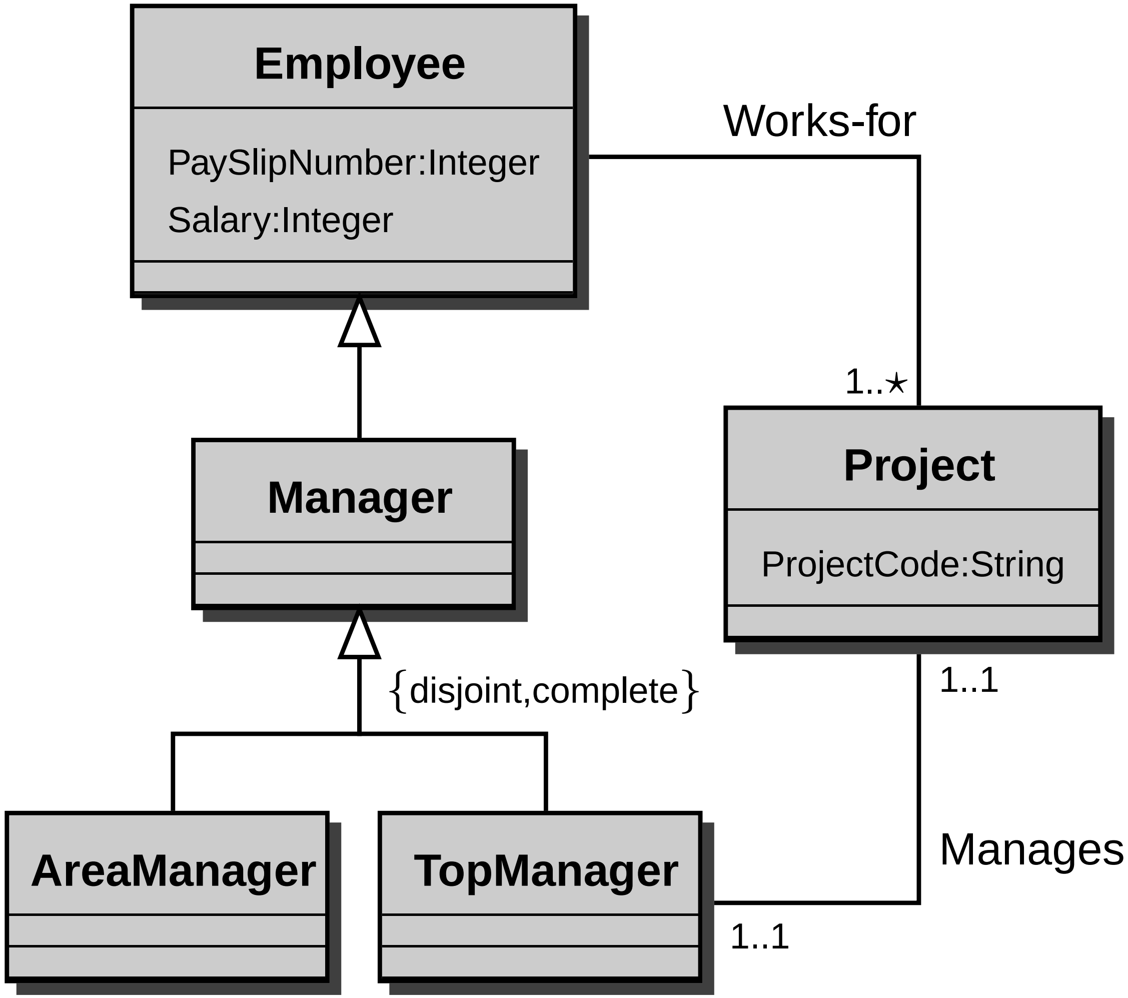
Ontology languages and Conceptual Data Models

- 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 and DAML+OIL).
- Ontology languages are typically expressed by means of diagrams.

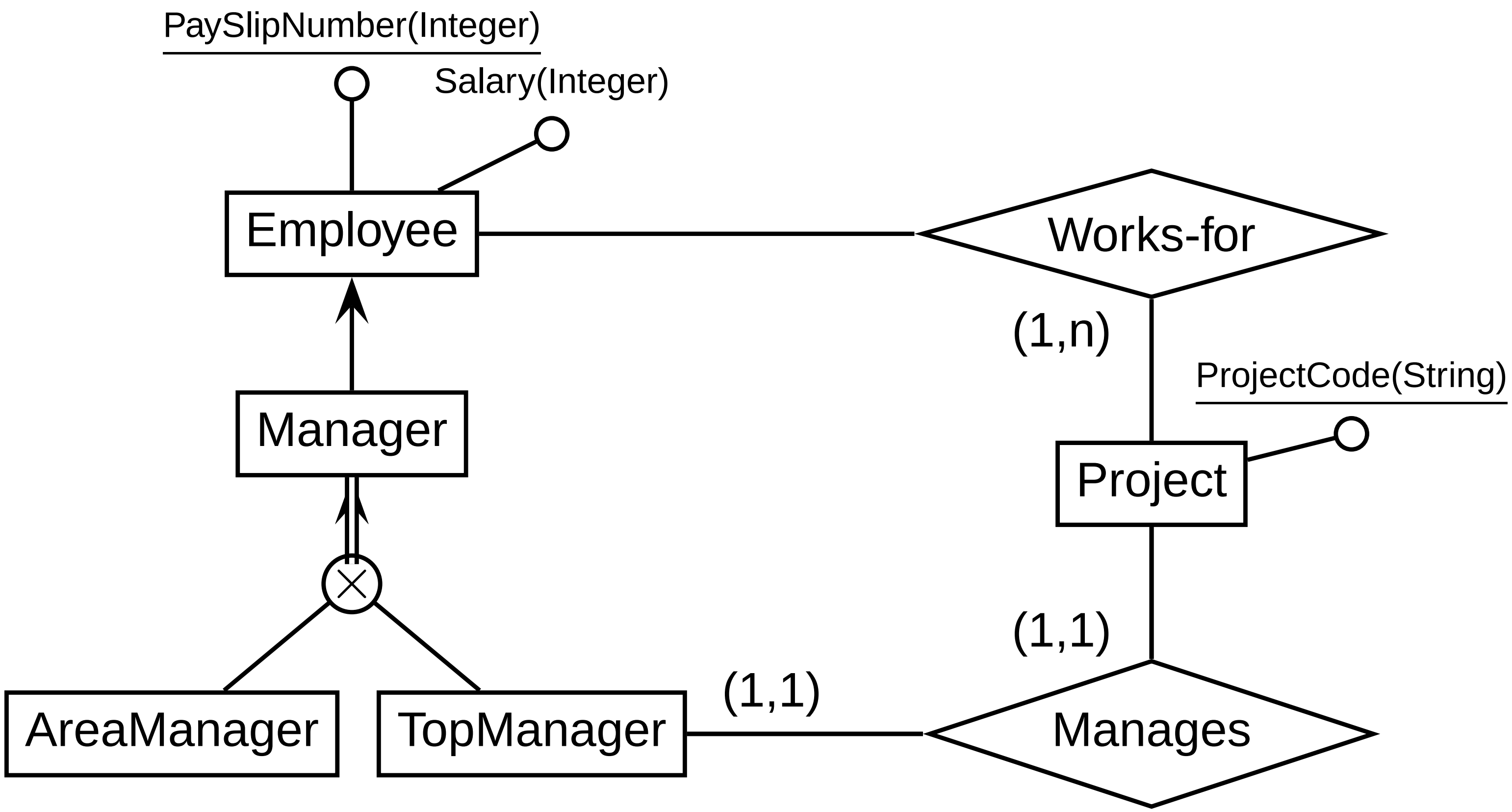
Ontology languages and Conceptual Data Models

- 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 and DAML+OIL).
- Ontology languages are typically expressed by means of diagrams.
- **Entity-Relationship** schemas and **UML** class diagrams can be considered as ontologies.

UML Class Diagram



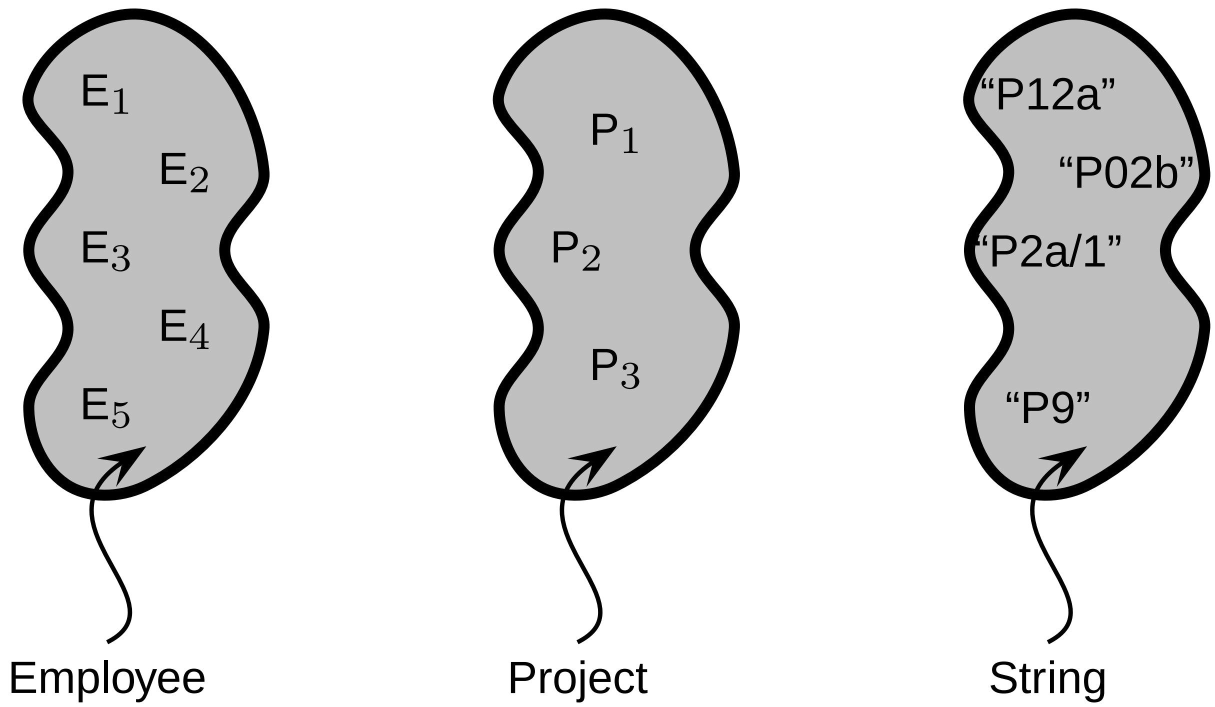
Entity-Relationship Schema



Semantics

In a specific world:

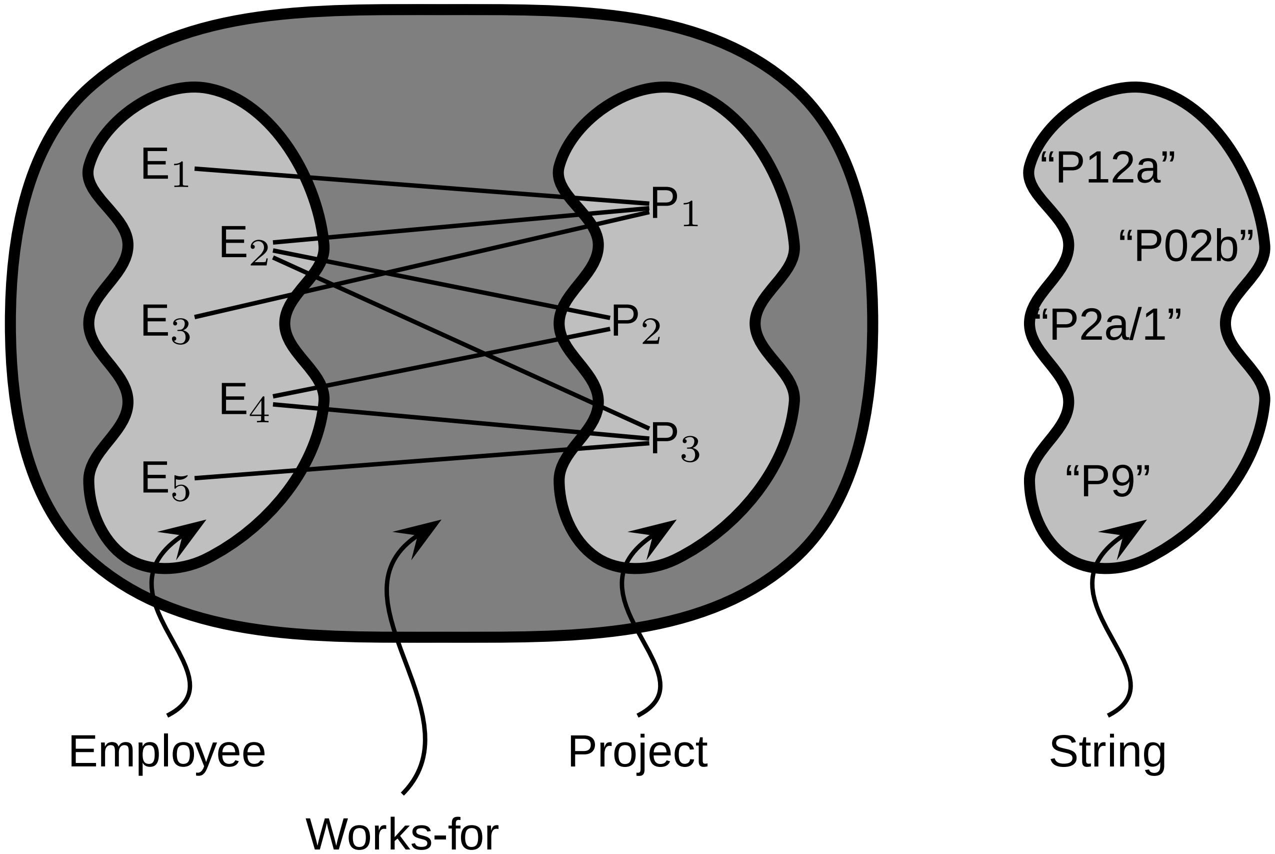
- A class is a ***set of instances***;
- a n-ary relationship is a ***set of n-tuples of instances***;
- an attribute is a ***set of pairs of an instance and a domain element***.



Semantics

In a specific world:

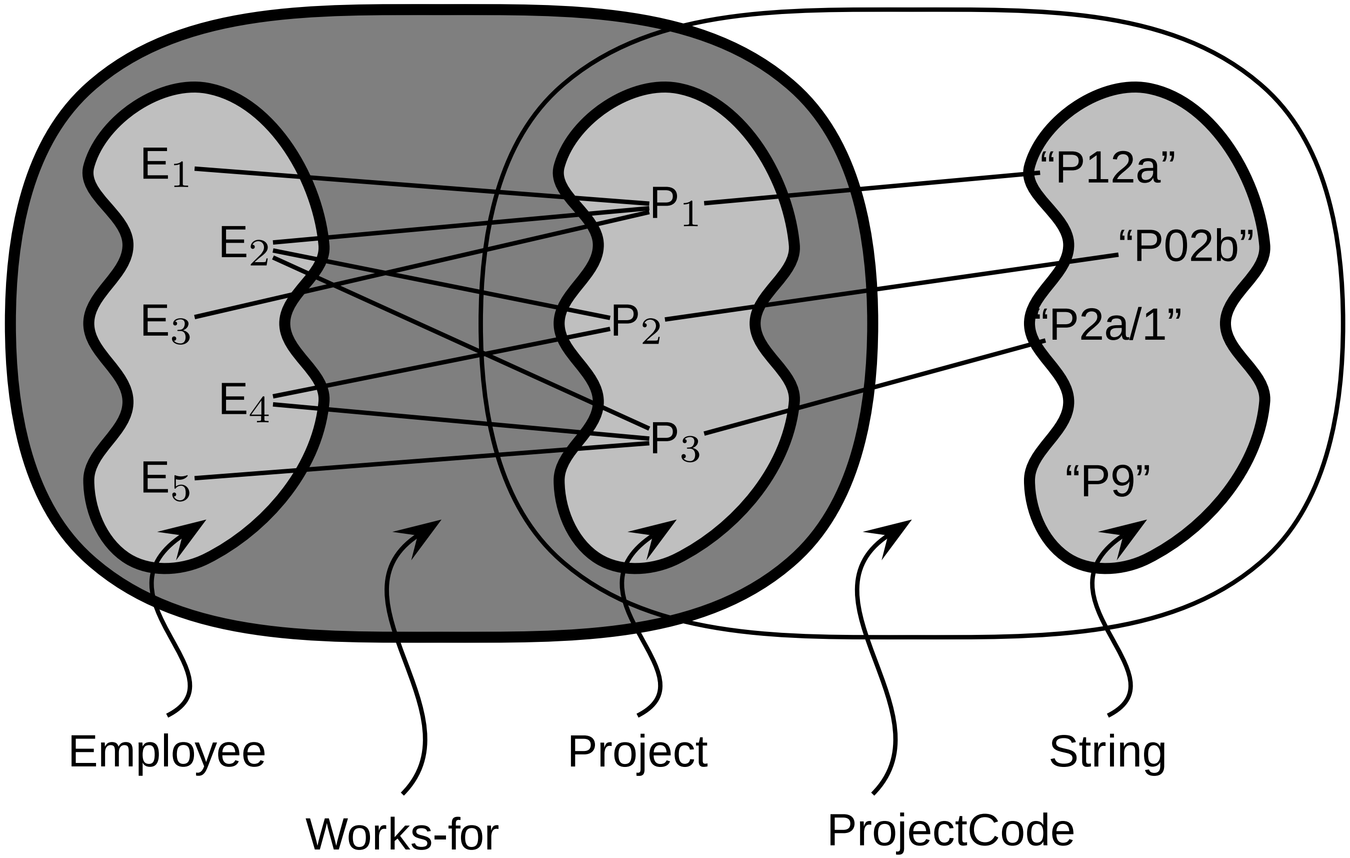
- A class is a ***set of instances***;
- a n-ary relationship is a ***set of n-tuples of instances***;
- an attribute is a ***set of pairs of an instance and a domain element***.



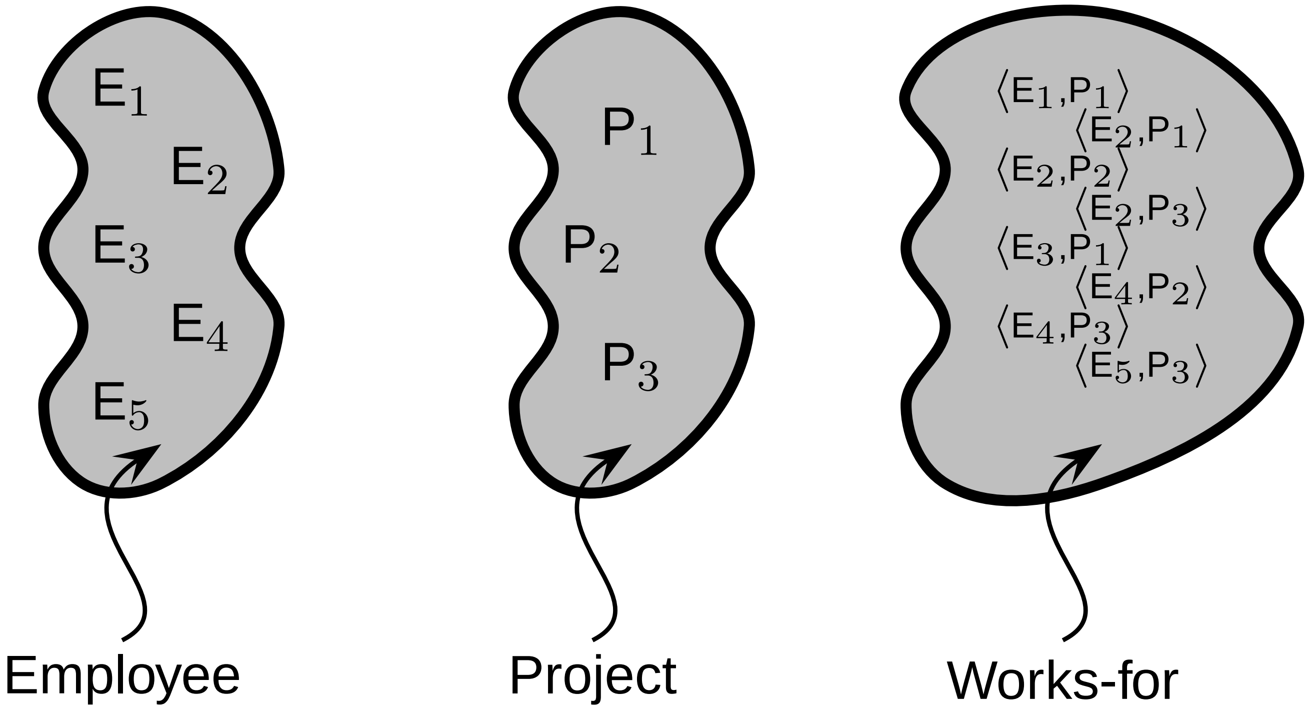
Semantics

In a specific world:

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



A world is described by sets of instances



The relational representation of a world

Employee

employeeId
E_1
E_2
E_3
E_4
E_5

Project

projectId
P_1
P_2
P_3

String

anystring
“P12a”
“P02b”
“P2a/1”
“P9”
...

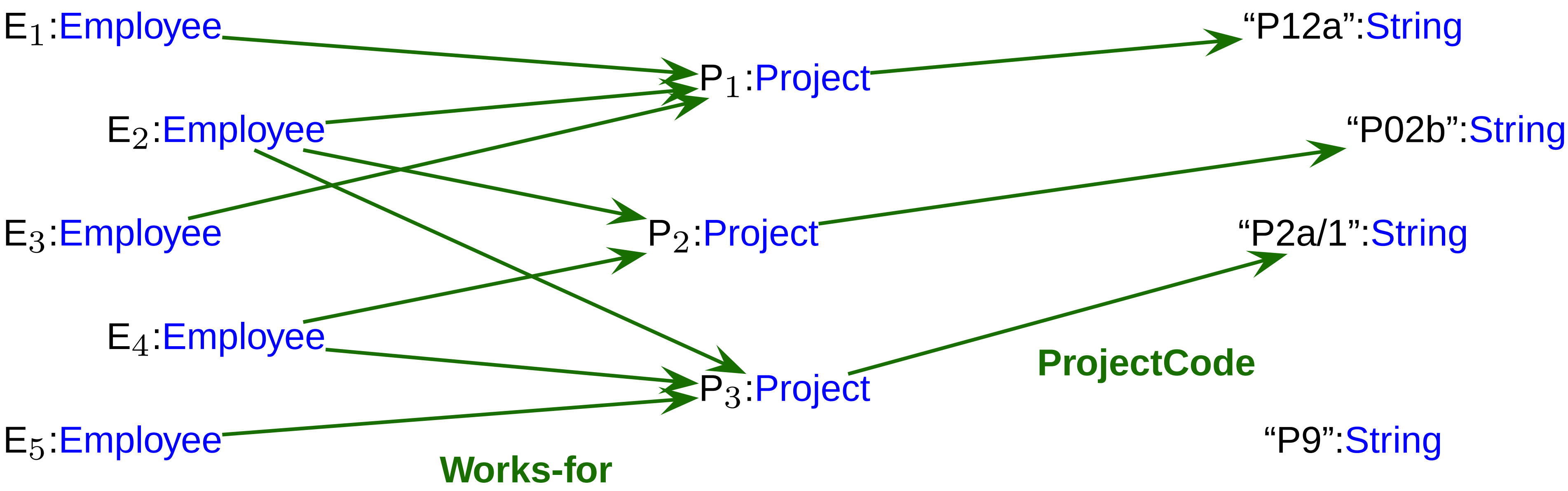
Works-for

employeeId	*projectId*
E_1	P_1
E_2	P_1
E_2	P_2
E_2	P_3
E_3	P_1
E_4	P_2
E_4	P_3
E_5	P_3

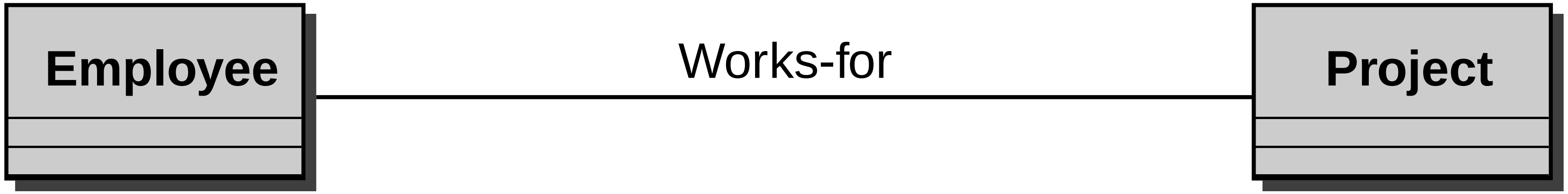
ProjectCode

projectId	*pcode*
P_1	“P12a”
P_2	“P02b”
P_3	“P2a/1”

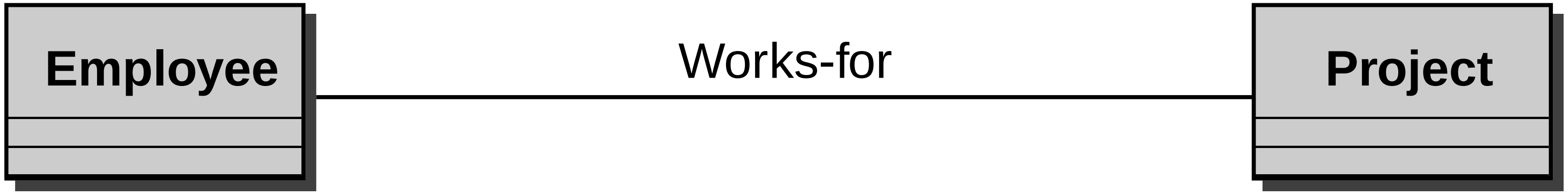
The graph representation of a world – e.g. RDF triples



Constraints introduced by Relationships

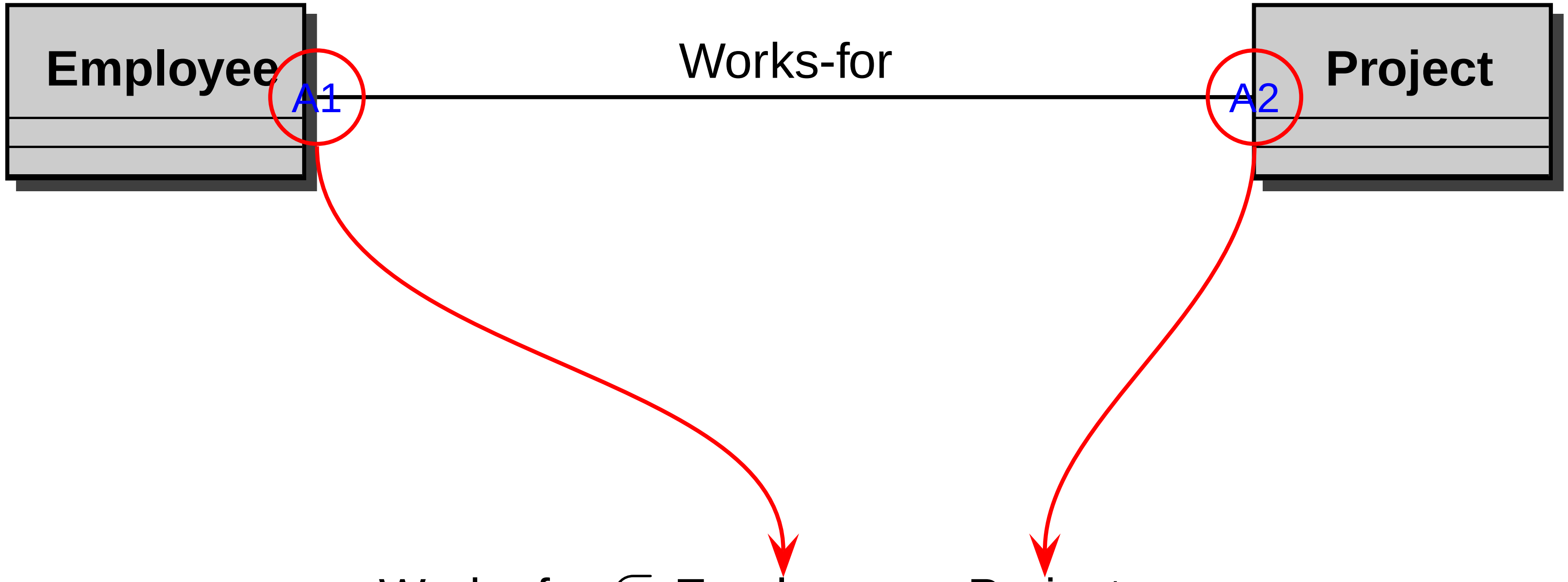


Constraints introduced by Relationships



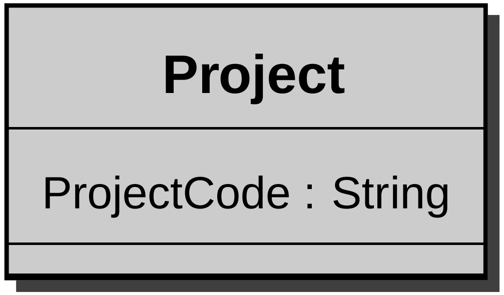
$$\text{Works-for} \subseteq \text{Employee} \times \text{Project}$$

Constraints introduced by Relationships

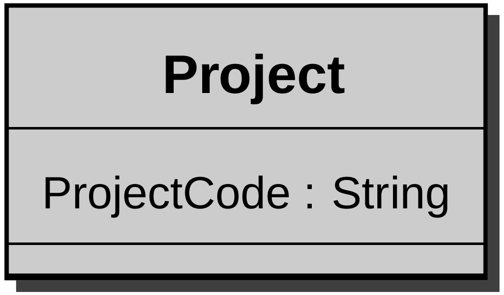


Works-for $\subseteq$ Employee $\times$ Project

Constraints introduced by Attributes

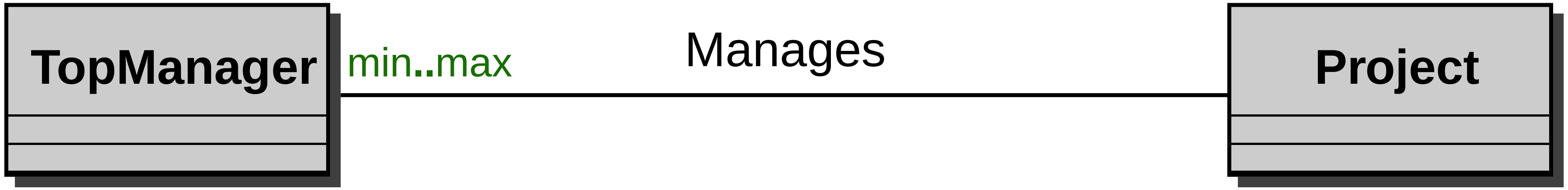


Constraints introduced by Attributes

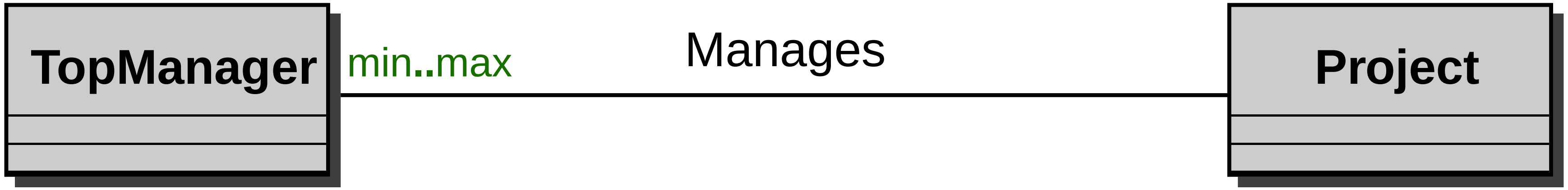


$$\text{Project} \subseteq \{p \mid \sharp(\text{ProjectCode} \cap (\{p\} \times \texttt{String})) \geq 1\}$$

Constraints introduced by Cardinality Constraints



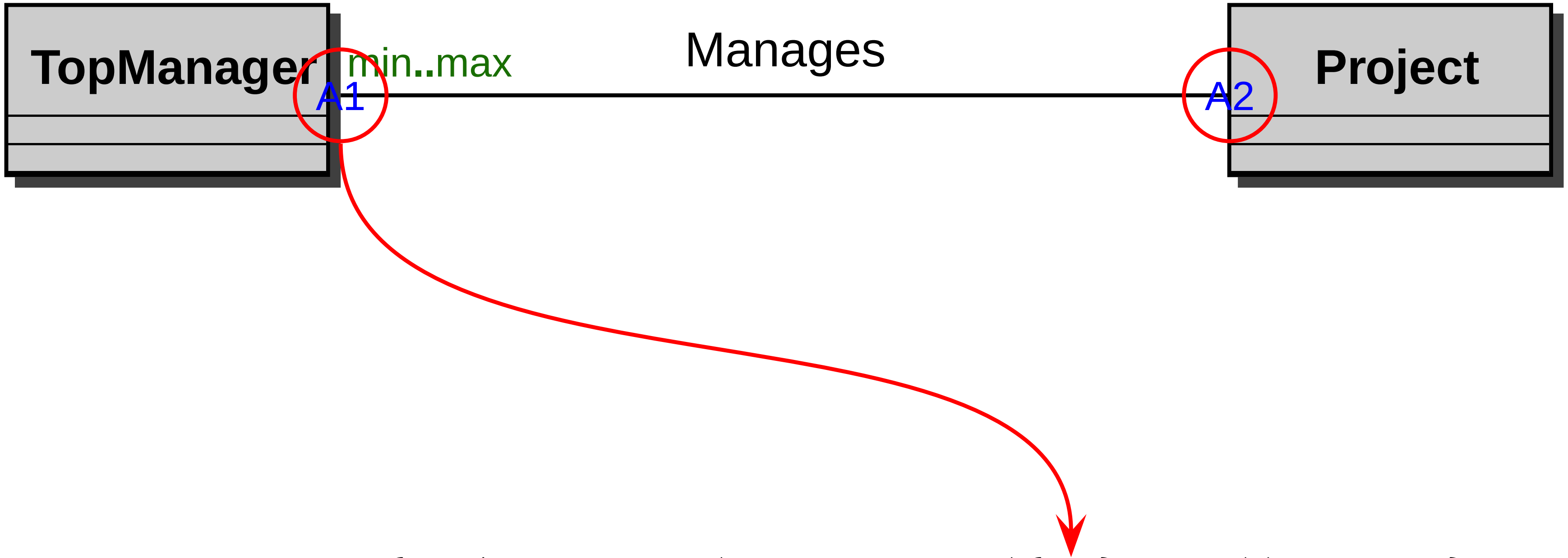
Constraints introduced by Cardinality Constraints



$$\text{TopManager} \subseteq \{m \mid \text{max} \geq \sharp(\text{Manages} \cap (\{m\} \times \Omega)) \geq \text{min}\}$$

(where Ω is the set of all instances)

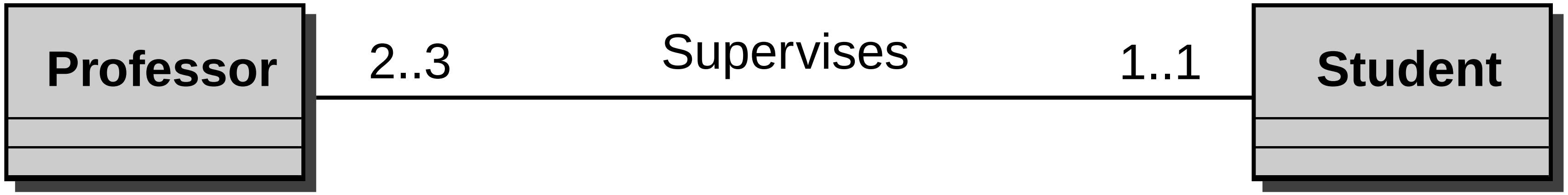
Constraints introduced by Cardinality Constraints



$$\text{TopManager} \subseteq \{m \mid \text{max} \geq \sharp(\text{Manages} \cap (\{m\} \times \Omega)) \geq \text{min}\}$$

(where Ω is the set of all instances)

The Cardinality Construct: An Example



A valid Database is:

Professor

professorId
Alex
Bob

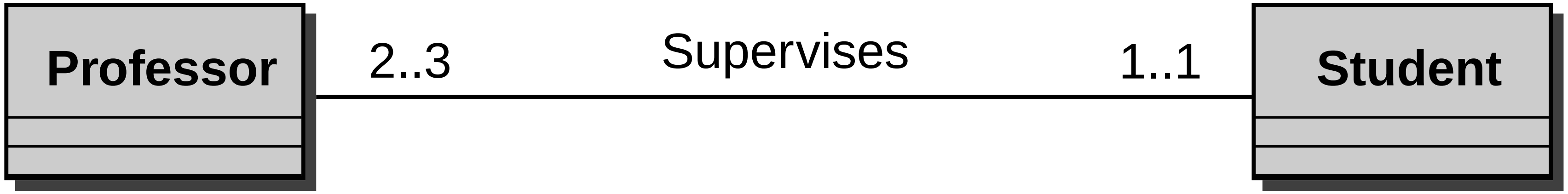
Student

studentId
John
Mary
Nick
Paul
Laura

Supervises

professorId	*studentId*
Alex	John
Bob	Laura
Alex	Mary
Bob	Nick
Alex	Paul

The Cardinality Construct: An Example



An invalid Database is:

Professor

professorId
Alex
Bob

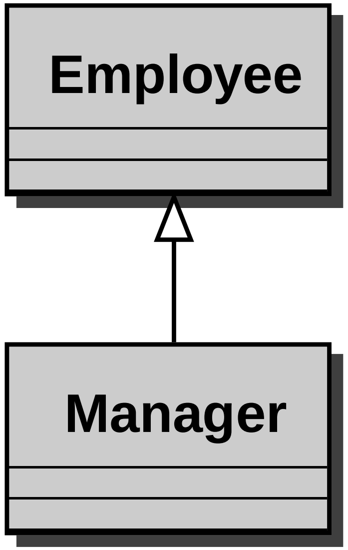
Student

studentId
John
Mary
Nick
Paul
Laura

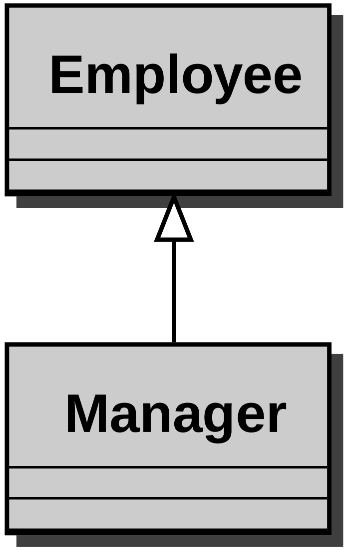
Supervises

professorId	*studentId*
Alex	John
Bob	Laura
Alex	Mary
Bob	Nick
Alex	Paul
Alex	Laura

Constraints introduced by ISA

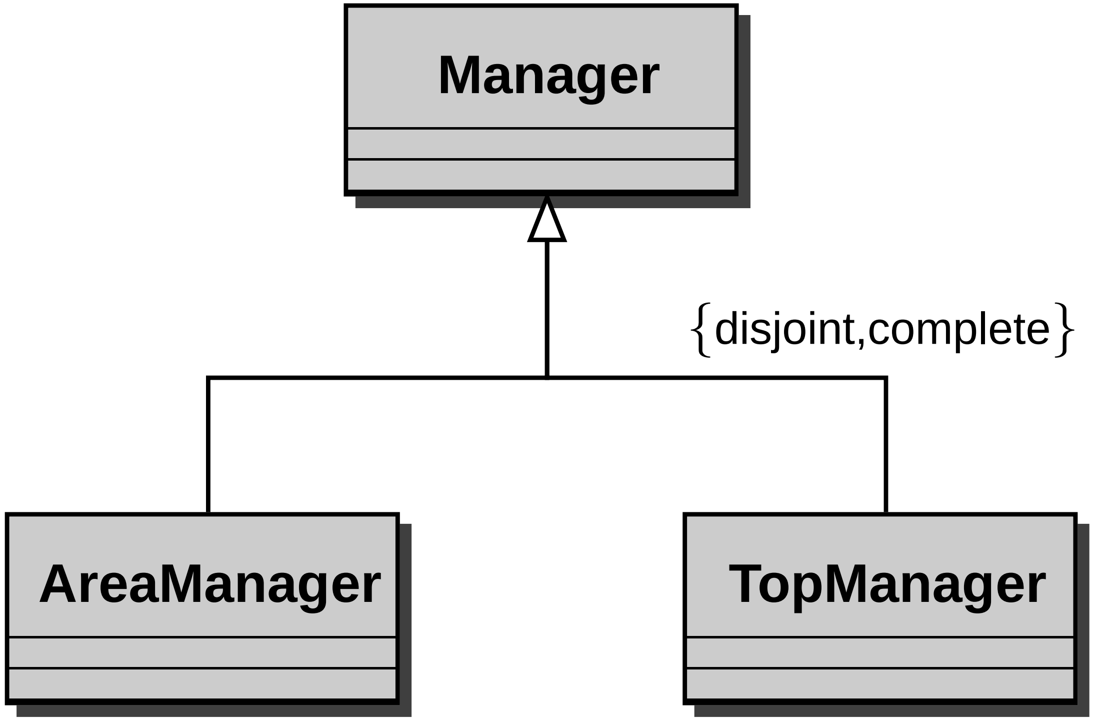


Constraints introduced by ISA

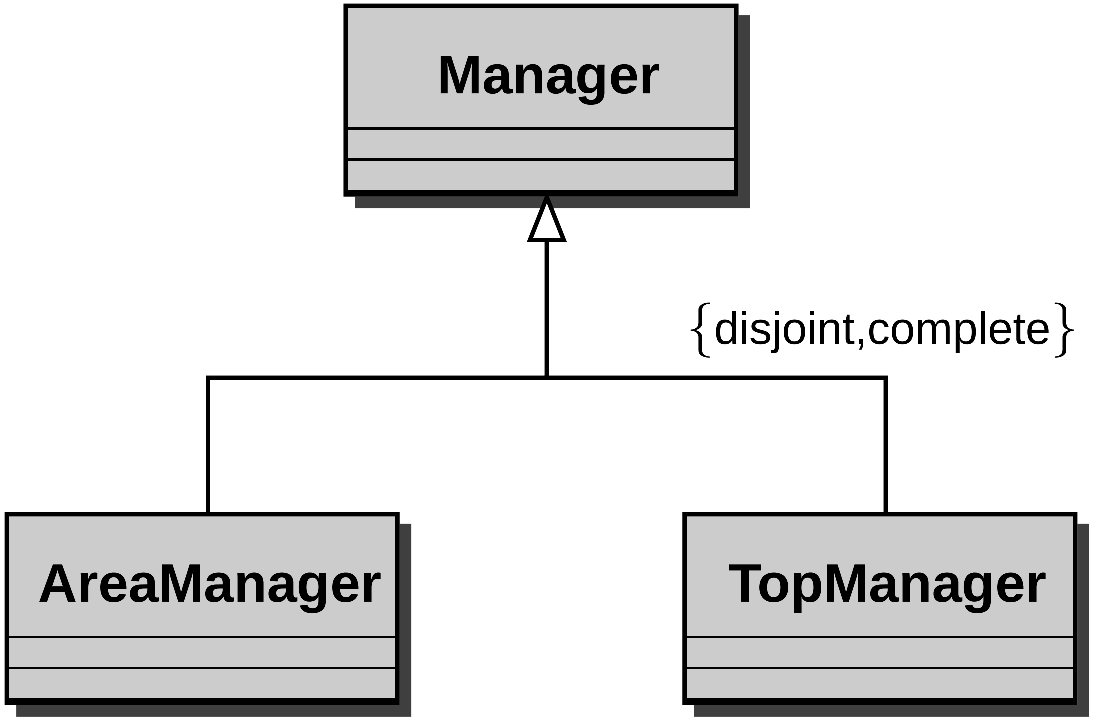


Manager $\subseteq$ Employee

Disjoint and *Total* constraints



Disjoint and *Total* constraints



- *ISA:* AreaManager $\subseteq$ Manager
- *ISA:* TopManager $\subseteq$ Manager
- *disjoint:* AreaManager $\cap$ TopManager $= \emptyset$
- *total:* Manager $\subseteq$ AreaManager $\cup$ TopManager

Constraints introduced by the initial diagram

Works-for $\subseteq$ Employee $\times$ Project

Manages $\subseteq$ TopManager $\times$ Project

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

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

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

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

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

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

Manager $\subseteq$ Employee

AreaManager $\subseteq$ Manager

TopManager $\subseteq$ Manager

AreaManager $\cap$ TopManager $= \emptyset$

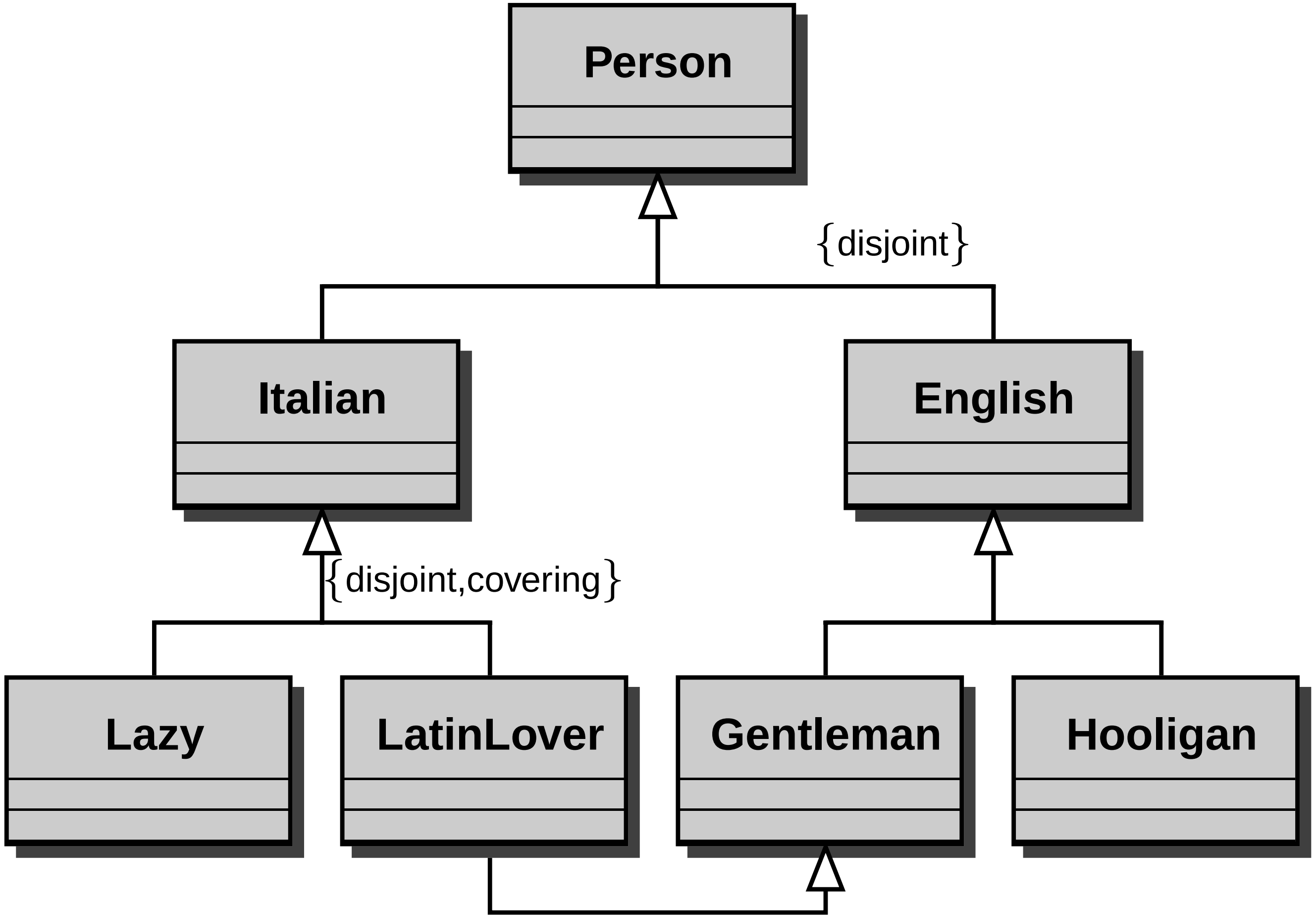
Manager $\subseteq$ AreaManager $\cup$ TopManager

Reasoning

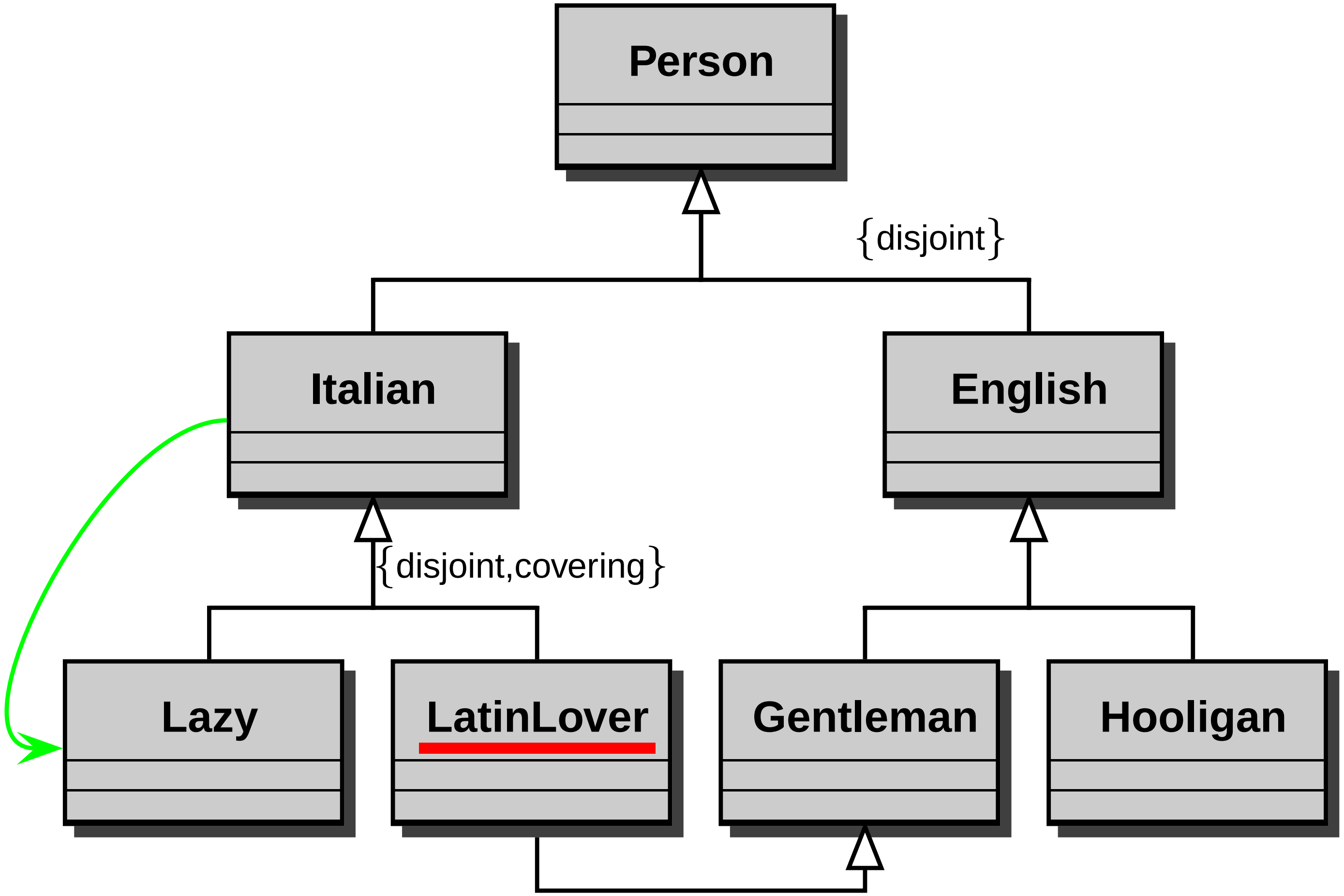
Given an ontology – seen as a collection of constraints – it is possible that additional constraints can be inferred.

- 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



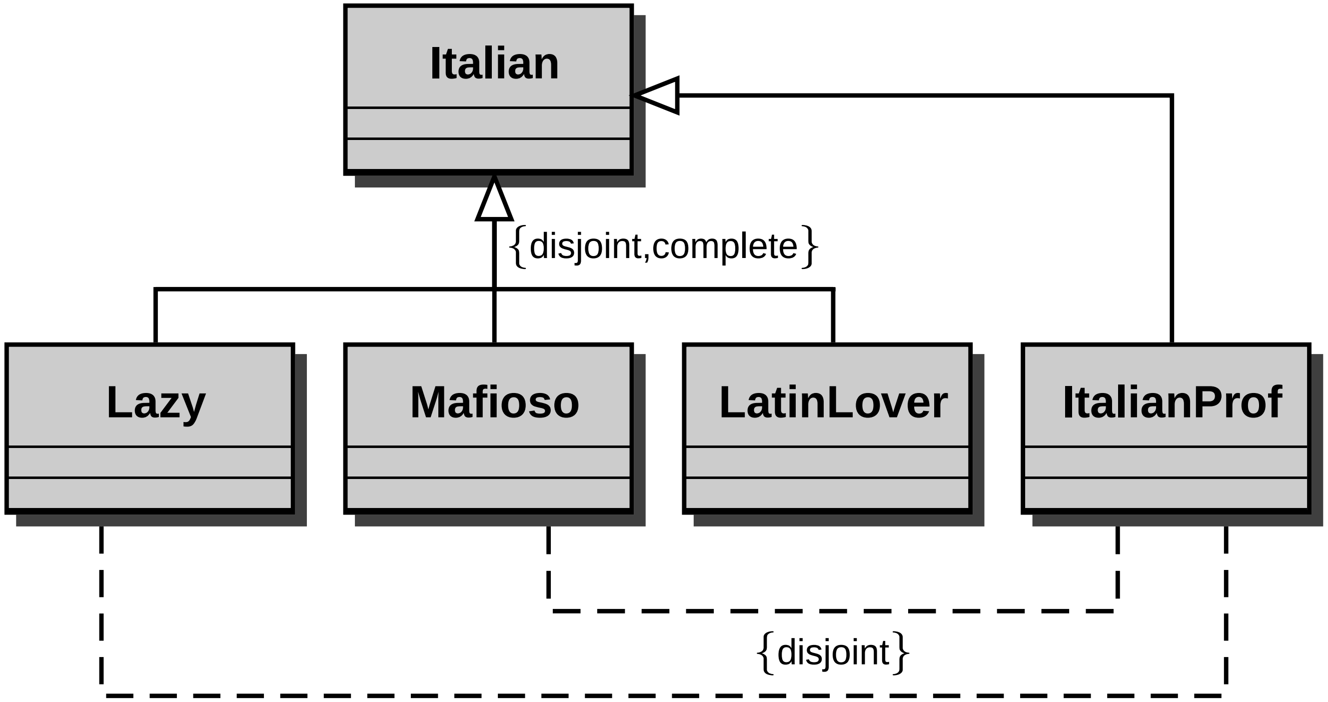
implies

LatinLover $= \emptyset$

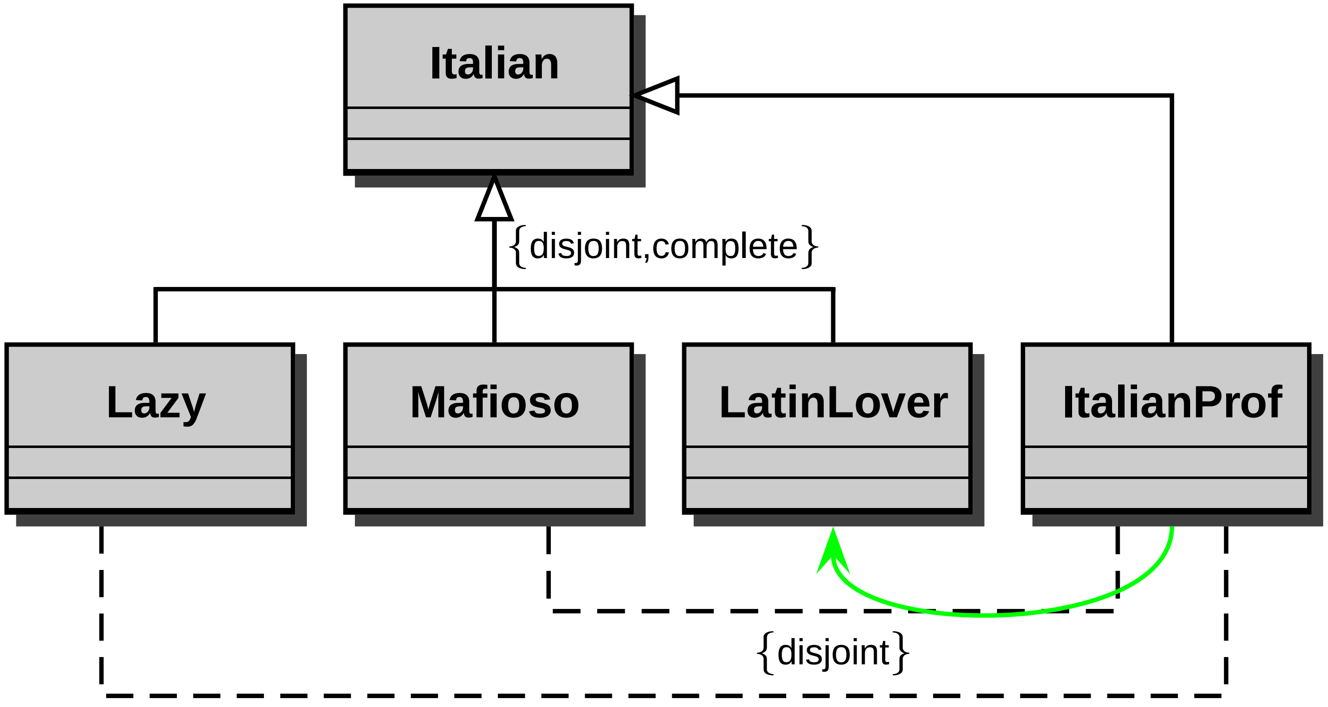
Italian $\subseteq$ Lazy

Italian $\equiv$ Lazy

Reasoning by cases



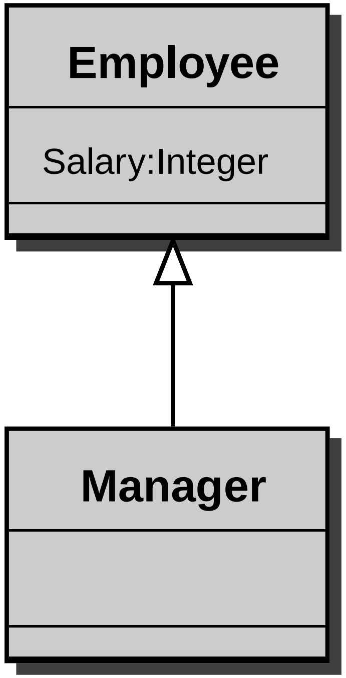
Reasoning by cases



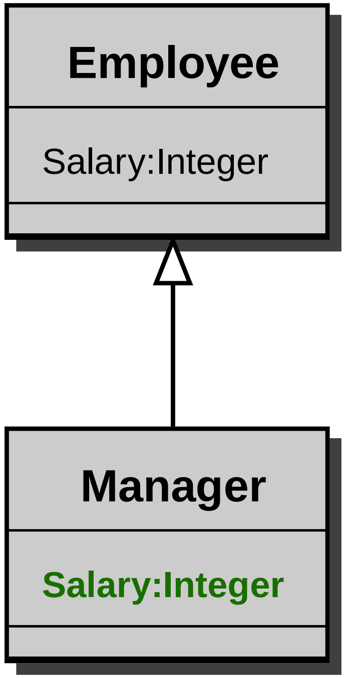
implies

ItalianProf $\subseteq$ LatinLover

ISA and Inheritance



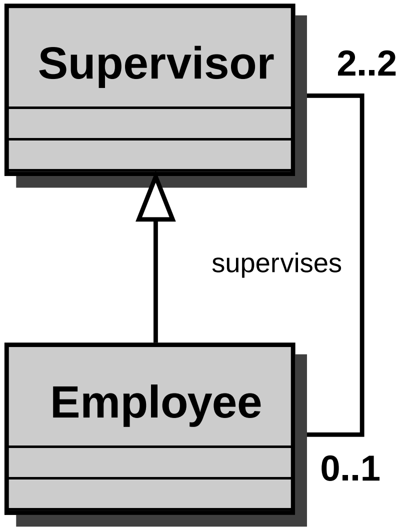
ISA and Inheritance



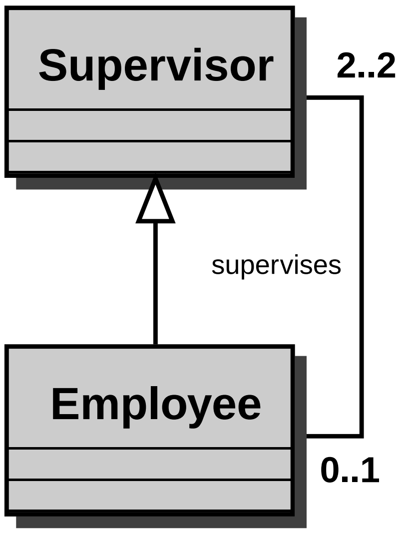
implies

$$\text{Manager} \subseteq \{m \mid \sharp(\text{Salary} \cap (\{m\} \times \texttt{Integer})) \geq 1\}$$

Infinite worlds



Infinite worlds

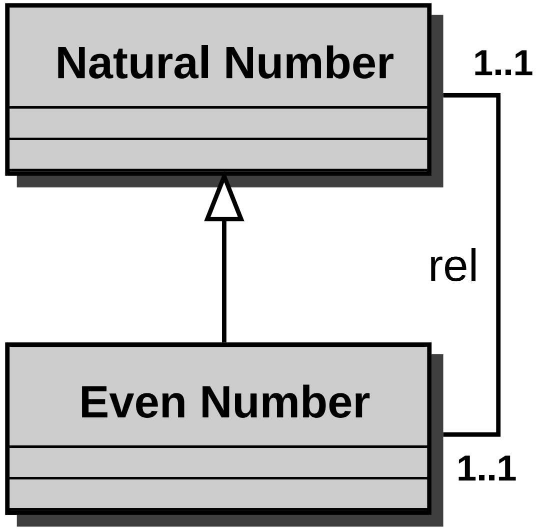


implies

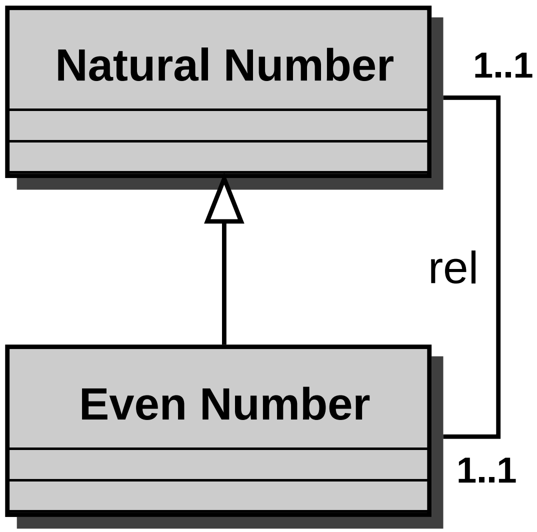
“the classes Employee and Supervisor contain an infinite number of instances”.

Therefore, the schema is inconsistent.

Bijection



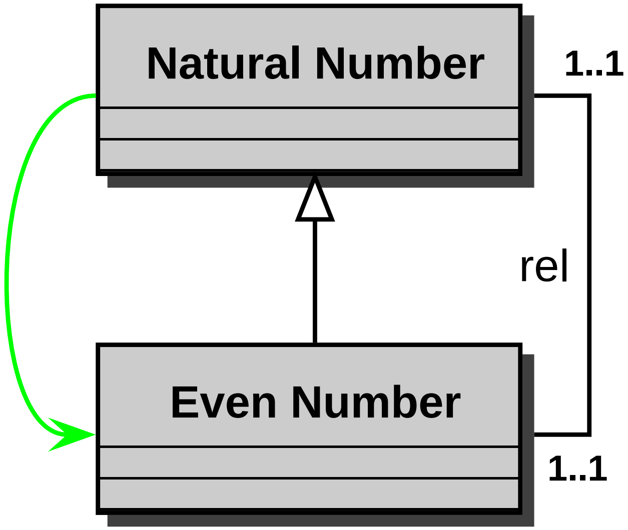
Bijection



implies

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

Bijection



implies

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

If the domain is finite: Natural Number $\equiv$ Even Number

i●com: Intelligent Conceptual Modelling tool

- i●com allows for the specification of multiple UML (or EER) diagrams and inter- and intra-schema constraints;
- Complete logical reasoning is employed by the tool using a hidden underlying (description logic) inference engine;
- i●com verifies the specification, infers implicit facts and stricter constraints, and manifests any inconsistencies during the conceptual modelling phase.
- `www.cs.man.ac.uk/~franconi/icom/`

Ontologies in First Order Logic

- We have introduced ontology languages that specify a set of constraints that should be satisfied by the world of interest.
- The *interpretation* of an ontology is therefore defined as the collection of all the *legal world descriptions* – i.e., all the (finite) relational structures which conform to the constraints imposed by the ontology.
- An alternative way to define the interpretation: an ontology is mapped into a set of First Order Logic (FOL) formulas.
- The legal world descriptions (i.e., the interpretation) of an ontology are all the models of the FOL theory associated to it.

FOL: The Alphabet

The Alphabet of the FOL language will have the following set of *Predicate* symbols:

- 1-ary predicate symbols: $E_1, E_2, \ldots, E_n$ for each Class (Entity); $D_1, D_2, \ldots, D_m$ for each Basic Domain.
- binary predicate symbols: $A_1, A_2, \ldots, A_k$ for each Attribute.
- n-ary predicate symbols: $R_1, R_2, \ldots, R_p$ for each Association (Relation).

FOL Notation

- *Vector variables* indicated as $\overline{x}$ stand for an n-tuple of variables:

  $\overline{x} = x_1, \ldots, x_n$

- *Counting existential quantifier* indicated as $\exists^{\leq n}$ or $\exists^{\geq n}$.

$$
\begin{aligned}
\exists^{\leq n} x.\, \varphi(x) \equiv \\
& \forall x_1, \ldots, x_n, x_{n+1}.\, \varphi(x_1) \wedge \ldots \wedge \varphi(x_n) \wedge \varphi(x_{n+1}) \rightarrow \\
& \qquad (x_1 = x_2) \vee \ldots \vee (x_1 = x_n) \vee (x_1 = x_{n+1}) \vee \\
& \qquad (x_2 = x_3) \vee \ldots \vee (x_2 = x_n) \vee (x_2 = x_{n+1}) \vee \\
& \qquad \ldots\ldots \vee (x_n = x_{n+1})
\end{aligned}
$$

$$
\begin{aligned}
\exists^{\geq n} x.\, \varphi(x) \equiv \\
& \exists x_1, \ldots, x_n.\, \varphi(x_1) \wedge \ldots \wedge \varphi(x_n) \wedge \\
& \qquad \neg(x_1 = x_2) \wedge \ldots \wedge \neg(x_1 = x_n) \wedge \\
& \qquad \neg(x_2 = x_3) \wedge \ldots \wedge \neg(x_2 = x_n) \wedge \\
& \qquad \ldots\ldots \wedge (x_{n-1} = x_n)
\end{aligned}
$$

The Interpretation function

Interpretation: $\mathcal{I} = \langle \mathbf{D}, \cdot^{\mathcal{I}} \rangle$, where $\mathbf{D}$ is an arbitrary non-empty set such that:

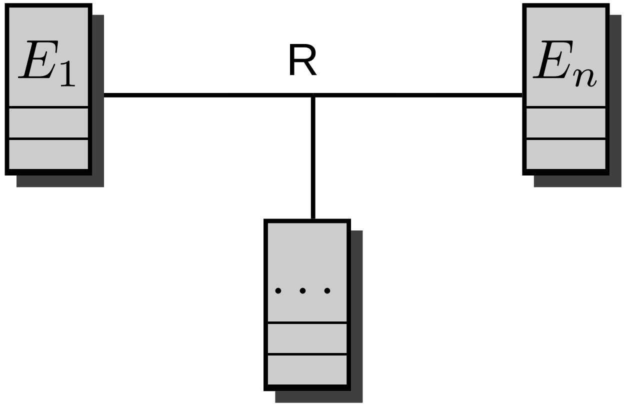
- $\mathbf{D} = \Omega \cup \mathcal{B}$, where:
  - $\mathcal{B} = \cup_{i=1}^{m} \mathcal{B}_{Di}$. $\mathcal{B}_{Di}$ is the set of values associated with each basic domain (i.e., integer, string, etc.); and $\mathcal{B}_{Di} \cap \mathcal{B}_{Dj} = \emptyset$, $\forall i, j \,\textbf{.}\, i \neq j$
  - Ω is the abstract entity domain such that $\mathcal{B} \cap \Omega = \emptyset$.

The Formal Semantics for the Atoms

$\mathcal{I}$ is the interpretation function that maps:

- *Basic Domain Predicates* to elements of the relative basic domain:
  ${D_i}^{\mathcal{I}} = \mathcal{B}_{Di}$ (e.g., *String*$^{\mathcal{I}} = \mathcal{B}_{\textit{String}}$).
- *Entity-set Predicates* to elements of the entity domain:
  ${E_i}^{\mathcal{I}} \subseteq \Omega$.
- *Attribute Predicates* to binary relations such that:
  ${A_i}^{\mathcal{I}} \subseteq \Omega \times \mathcal{B}$.
- *Relationship-set Predicates* to n-ary relations over the entity domain:
  ${R_i}^{\mathcal{I}} \subseteq \Omega \times \Omega \ldots \times \Omega = \Omega^n$.

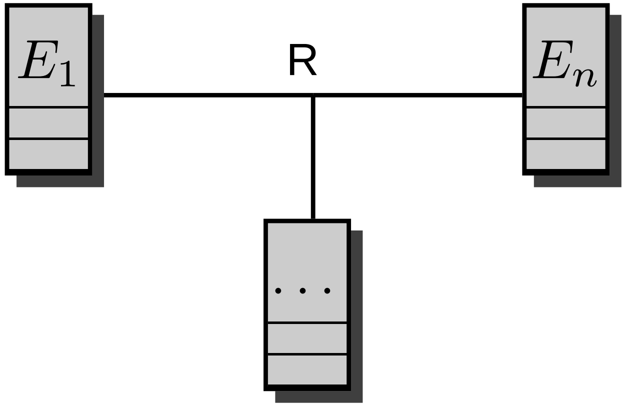
The Relationship Construct



- The meaning of this constraint is:

$$R^{\mathcal{I}} \subseteq {E_1}^{\mathcal{I}} \times \ldots \times {E_n}^{\mathcal{I}}$$

The Relationship Construct



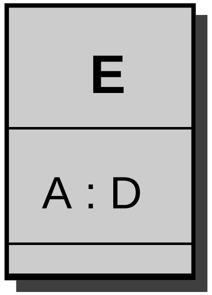
- The meaning of this constraint is:

$$R^{\mathcal{I}} \subseteq {E_1}^{\mathcal{I}} \times \ldots \times {E_n}^{\mathcal{I}}$$

- The FOL translation is the formula:

$$\forall x_1, \ldots, x_n . R(x_1, \ldots, x_n) \rightarrow E_1(x_1) \wedge \ldots \wedge E_n(x_n)$$

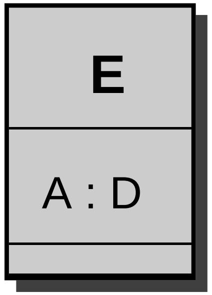
The Attribute Construct



- The meaning of this constraint is:

$$E^{\mathcal{I}} \subseteq \{e \in \Omega \mid \sharp(A^{\mathcal{I}} \cap (\{e\} \times \mathcal{B}_D)) \geq 1\}$$

The Attribute Construct



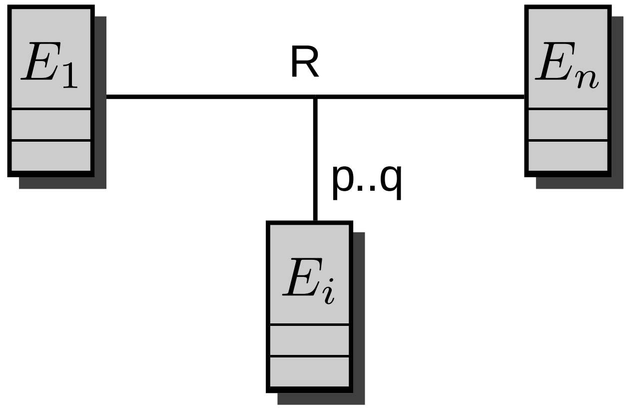
- The meaning of this constraint is:

$$E^{\mathcal{I}} \subseteq \{e \in \Omega \mid \sharp(A^{\mathcal{I}} \cap (\{e\} \times \mathcal{B}_D)) \geq 1\}$$

- The FOL translation is the formula:

$$\forall x.\, E(x) \rightarrow \exists y. A(x, y) \wedge D(y)$$

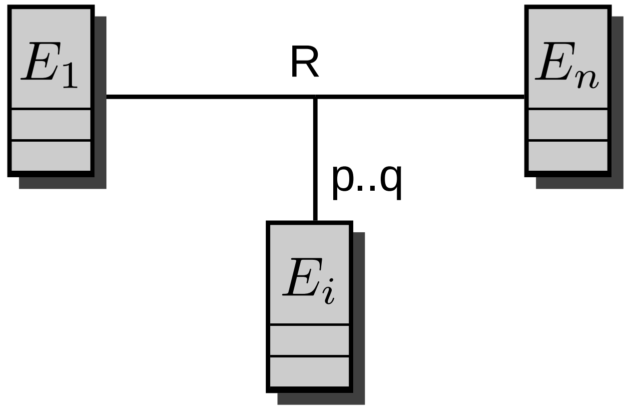
The Cardinality Construct



- The meaning of this constraint is:

$$E_i^{\mathcal{I}} \subseteq \{e_i \in \Omega \mid p \leq \sharp(R^{\mathcal{I}} \cap (\Omega \times \{e_i\} \times \Omega)) \leq q\}$$

The Cardinality Construct



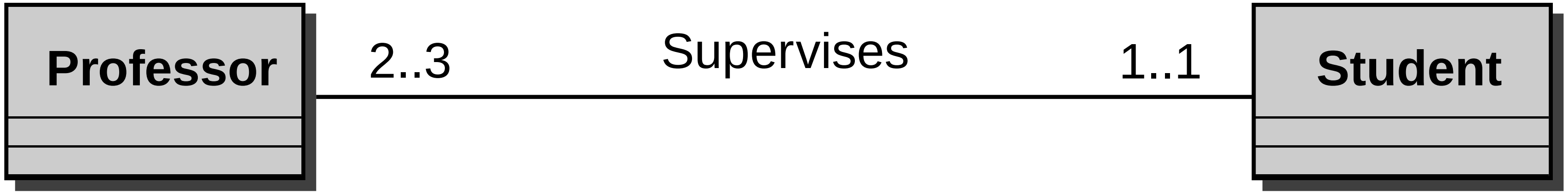
- The meaning of this constraint is:

$$E_i^{\mathcal{I}} \subseteq \{e_i \in \Omega \mid p \leq \sharp(R^{\mathcal{I}} \cap (\Omega \times \{e_i\} \times \Omega)) \leq q\}$$

- The FOL translation is the formula:

$$\begin{aligned} \forall x_i. E(x_i) \rightarrow & \exists^{\geq p} x_1, \ldots, x_{i-1}, x_{i+1}, \ldots x_n. R(x_1, \ldots, x_n) \land \\ & \exists^{\leq q} x_1, \ldots, x_{i-1}, x_{i+1}, \ldots x_n. R(x_1, \ldots, x_n) \end{aligned}$$

The Cardinality Construct: An Example



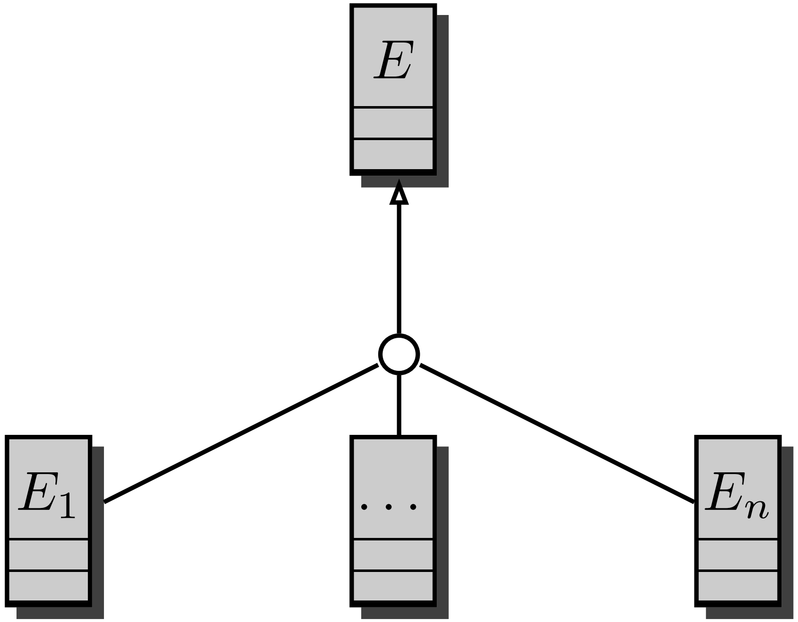
- The FOL translation is:

$$\forall x, y. \texttt{Supervises}(x, y) \rightarrow \texttt{Professor}(x) \wedge \texttt{Student}(y)$$

$$\forall x. \texttt{Professor}(x) \rightarrow \exists^{\geq 2} y. \texttt{Supervises}(x, y) \wedge \exists^{\leq 3} y. \texttt{Supervises}(x, y)$$

$$\forall y. \texttt{Student}(y) \rightarrow \exists^{=1} x. \texttt{Supervises}(x, y)$$

ISA Relations



- The meaning of this constraint is:

$$E_i^{\mathcal{I}} \subseteq E^{\mathcal{I}}, \text{ for all } i = 1, \dots, n.$$

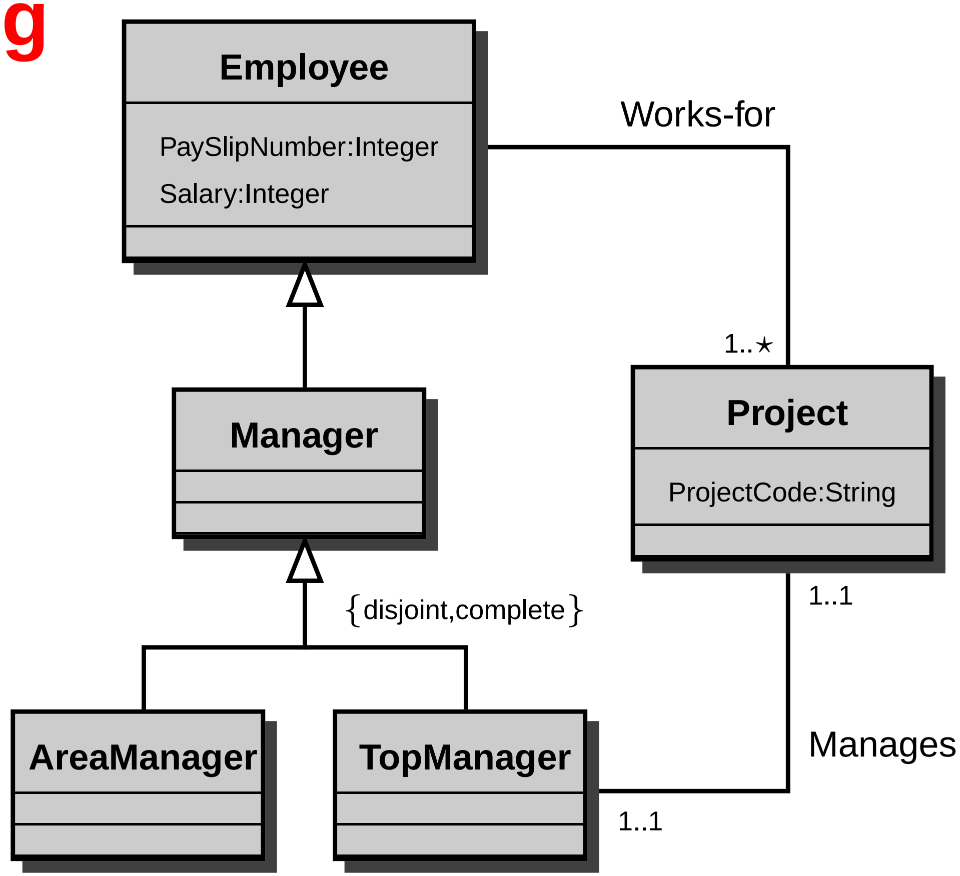
- The FOL translation is the formula:

$$\forall x. E_i(x) \rightarrow E(x), \text{ for all } i = 1, \dots, n.$$

Disjoint and covering constraints

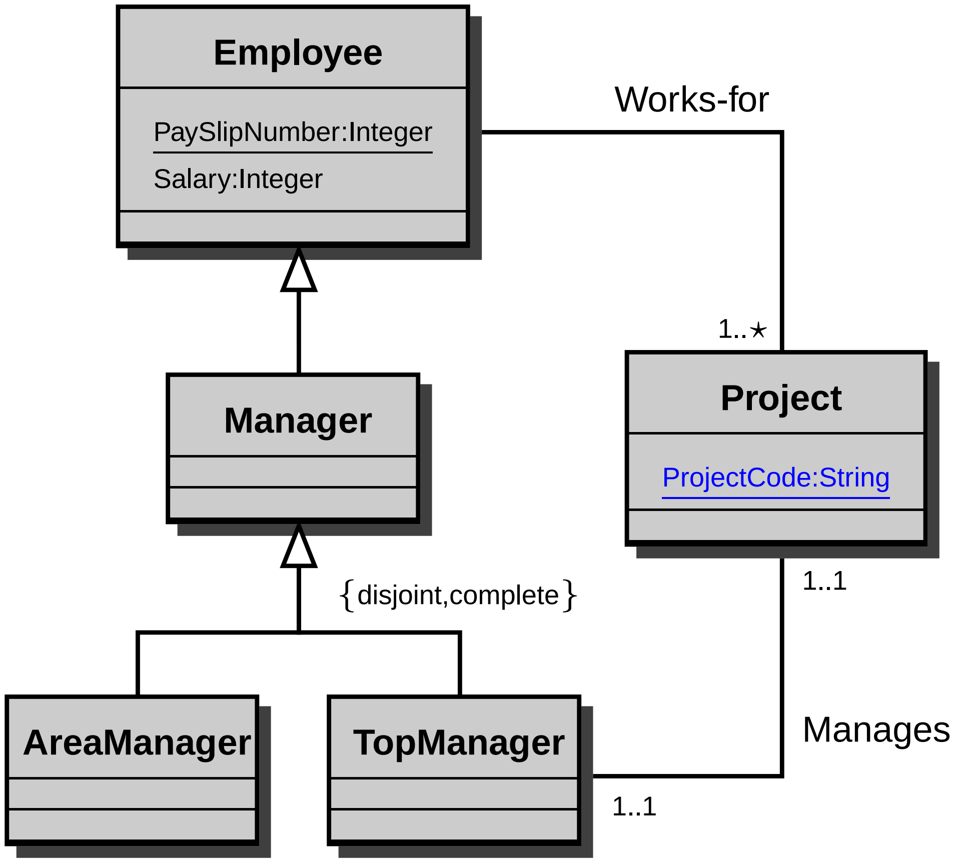
The encoding in FOL of disjoint and covering constraints is left as an exercise.

FOL encoding



$$
\begin{array}{lcl}
\forall x, y.\,\texttt{Works-for}(x,y) & \rightarrow & \texttt{Employee}(x) \land \texttt{Project}(y) \\
\forall x, y.\,\texttt{Manages}(x,y) & \rightarrow & \texttt{Top-Manager}(x) \land \texttt{Project}(y) \\
\forall y.\,\texttt{Project}(y) & \rightarrow & \exists x.\,\texttt{Works-for}(x,y) \\
\forall y.\,\texttt{Project}(y) & \rightarrow & \exists^{=1} x.\,\texttt{Manages}(x,y) \\
\forall x.\,\texttt{Top-Manager}(x) & \rightarrow & \exists^{=1} y.\,\texttt{Manages}(x,y) \\
\forall x.\,\texttt{Manager}(x) & \rightarrow & \texttt{Employee}(x) \\
\forall x.\,\texttt{Manager}(x) & \rightarrow & \texttt{Area-Manager}(x) \lor \texttt{Top-Manager}(x) \\
\forall x.\,\texttt{Area-Manager}(x) & \rightarrow & \texttt{Manager}(x) \land \neg\texttt{Top-Manager}(x) \\
\forall x.\,\texttt{Top-Manager}(x) & \rightarrow & \texttt{Manager}(x)
\end{array}
$$

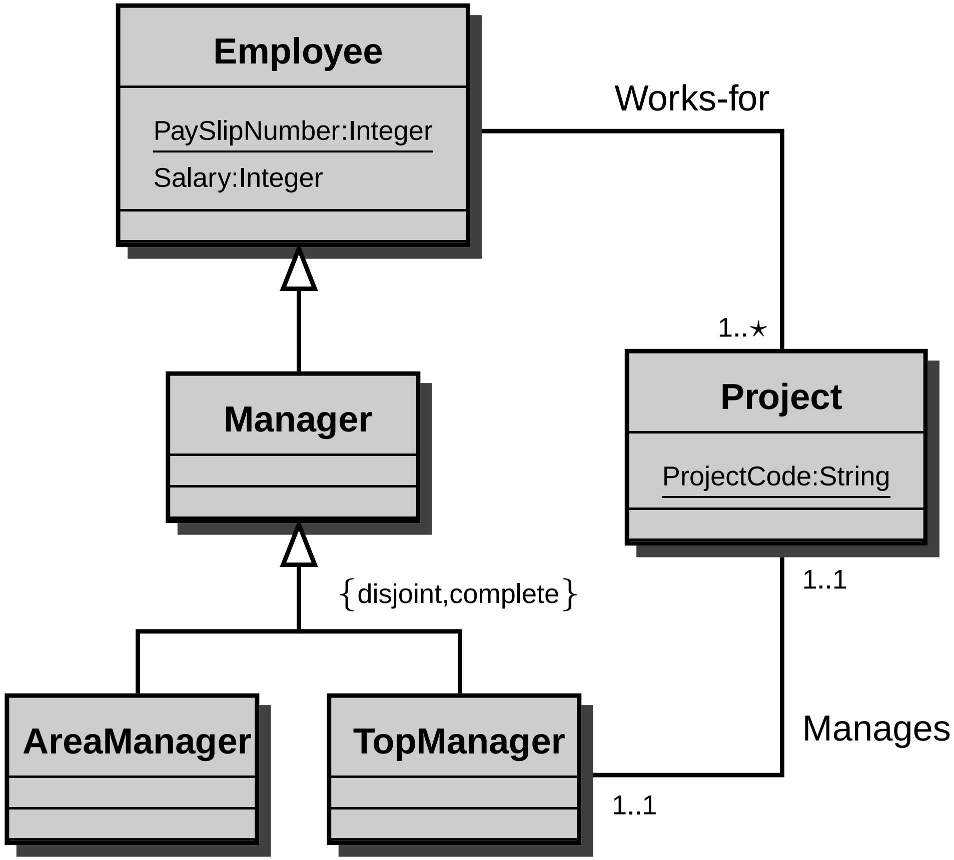
Key constraints



$$\forall x.\ \texttt{Project}(x) \rightarrow \exists^{=1} y.\ \texttt{ProjectCode}(x,y) \wedge \texttt{String}(y)$$

$$\forall y.\ \exists x.\ \texttt{ProjectCode}(x,y) \rightarrow \exists^{=1} x.\ \texttt{ProjectCode}(x,y) \wedge \texttt{Project}(x)$$

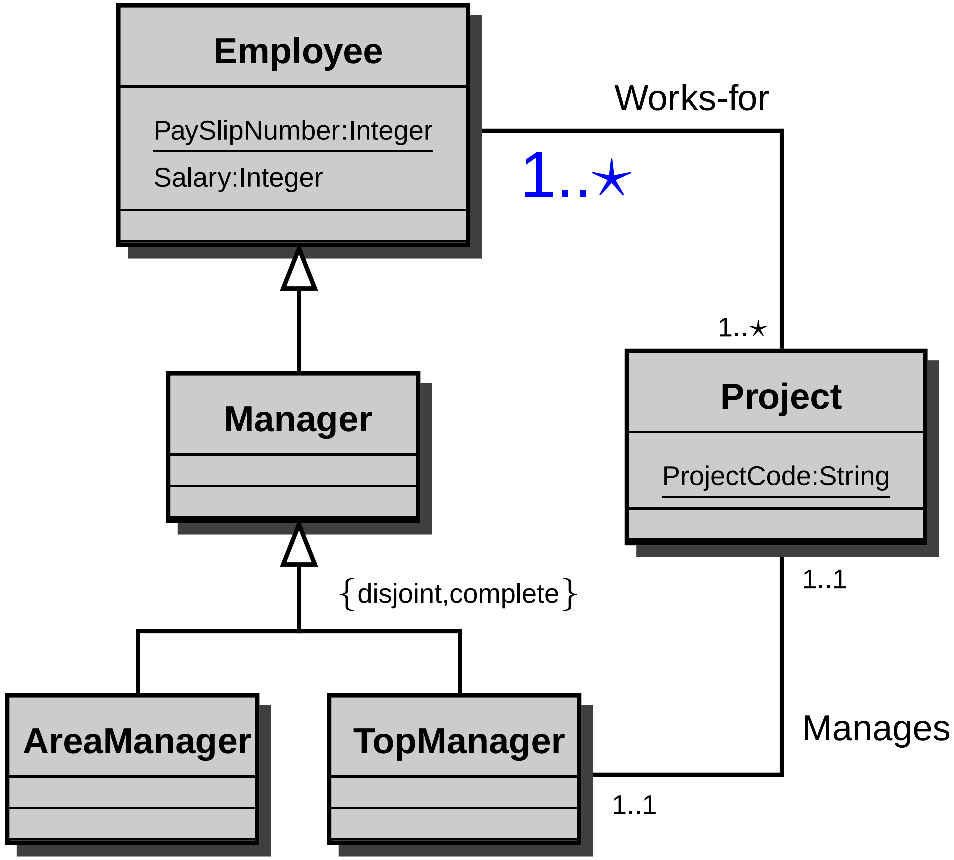
Additional constraints



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

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

Additional constraints



- 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

The $\mathcal{DLR}$ Description Logic – a fragment of FOL

- **relationships:** interpreted as **sets of tuples** of a given arity

$$R \rightarrow \top_n \mid RN \mid \neg R \mid R_1 \sqcap R_2 \mid R_1 \sqcup R_2 \mid i/n : C$$

- **classes:** interpreted as **sets of objects**

$$C \rightarrow \top \mid CN \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists^{\leqslant k}[i]R$$

- **conceptual schema**: $R \sqsubseteq R' \mid C \sqsubseteq C' \mid R \not\sqsubseteq R' \mid C \not\sqsubseteq C'$

$$\texttt{Works-for} \sqsubseteq \texttt{subj}/2 : \texttt{Employee} \sqcap \texttt{obj}/2 : \texttt{Project}$$

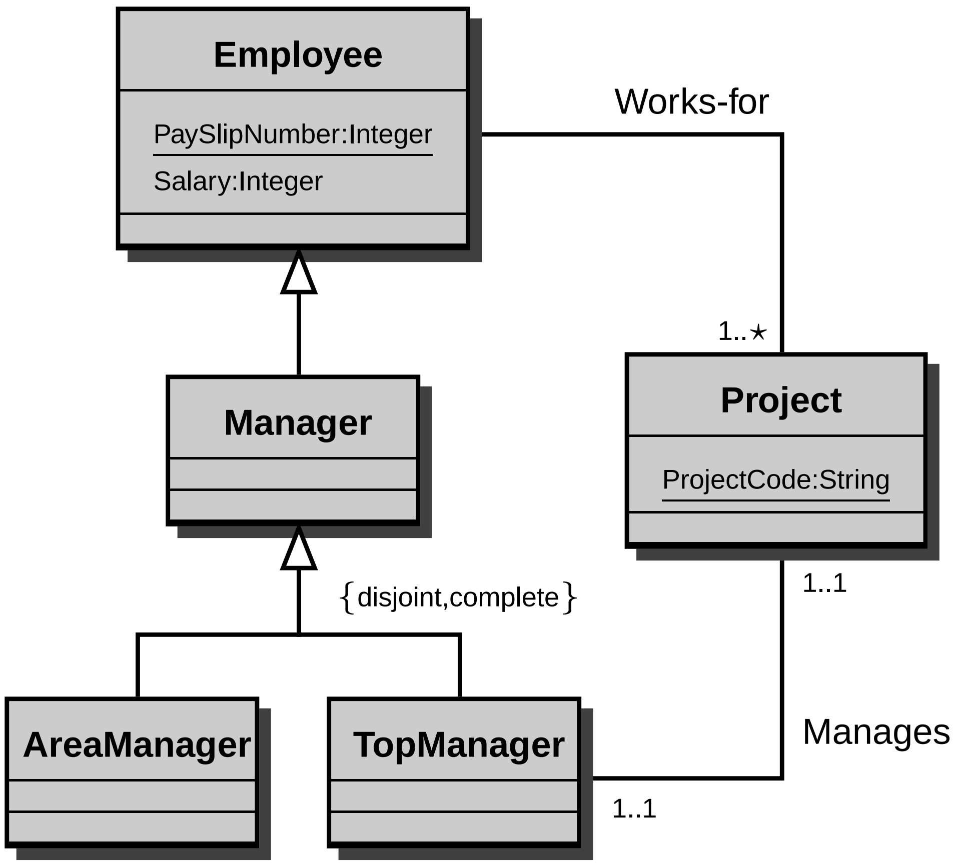
$$\texttt{TopManager} \sqsubseteq \texttt{Manager} \sqcap \exists^{=1}[\texttt{man}]\texttt{Manages}$$

Encoding ontologies in Description Logics

- Object-oriented data models (e.g., UML and ODMG)
- Semantic data models (e.g., EER and ORM)
- Frame-based ontology languages (e.g., DAML+OIL)

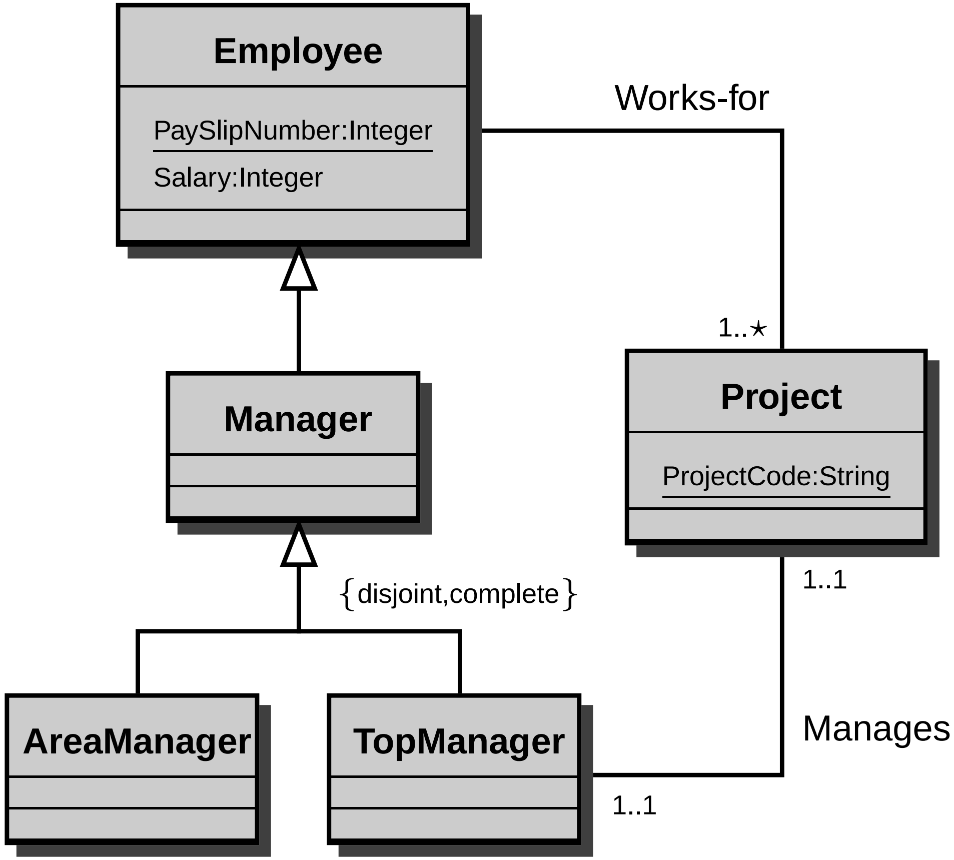
Encoding ontologies in Description Logics

- Object-oriented data models (e.g., UML and ODMG)
- Semantic data models (e.g., EER and ORM)
- Frame-based ontology languages (e.g., DAML+OIL)
- 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.



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

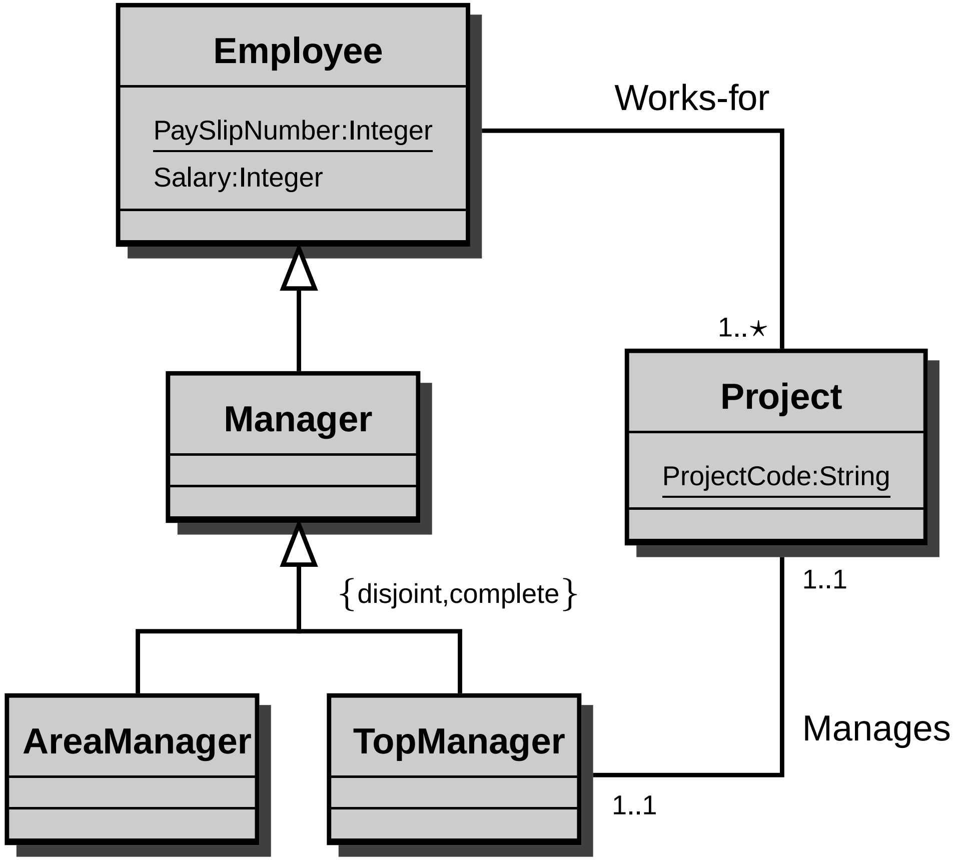
Deducing constraints



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):

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

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

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

Extensions of $\mathcal{DLR}$

- $\mathcal{DLR}_{reg}$: regular expressions and recursive views (beyond FOL)
- $\mathcal{DLR}_{\mathcal{US}}$: temporal constructs to model temporal databases (temporal logic)
- $\mathcal{DLR}_{key}$: general key constraints

Reasoning with Ontologies

- Exploit the $\mathcal{DLR}$ reasoning procedures for solving reasoning problems in the ontology enriched with constraints.
- Logical implication and consistency for $\mathcal{DLR}$ knowledge bases is decidable and EXPTIME-complete, and practical, proved correct and complete algorithms exist in implemented systems.

Reasoning with Ontologies

- Exploit the $\mathcal{DLR}$ reasoning procedures for solving reasoning problems in the ontology enriched with constraints.
- Logical implication and consistency for $\mathcal{DLR}$ knowledge bases is decidable and EXPTIME-complete, and practical, proved correct and complete algorithms exist in implemented systems.
- $\rightsquigarrow$ Ontology consistency checking with constraints and logical implication of constraints in ontologies are all decidable EXPTIME-complete problems.

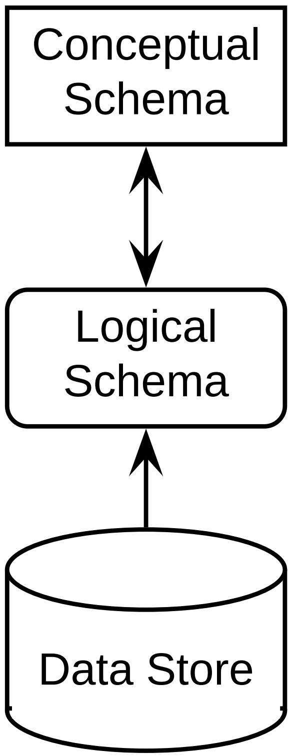
Reasoning with Ontologies

- Exploit the $\mathcal{DLR}$ reasoning procedures for solving reasoning problems in the ontology enriched with constraints.
- Logical implication and consistency for $\mathcal{DLR}$ knowledge bases is decidable and EXPTIME-complete, and practical, proved correct and complete algorithms exist in implemented systems.
- $\rightsquigarrow$ Ontology consistency checking with constraints and logical implication of constraints in ontologies are all decidable EXPTIME-complete problems.
- i●com is an implemented conceptual modelling tool using in the background a $\mathcal{DLR}$ ontology server supporting the ontology design.

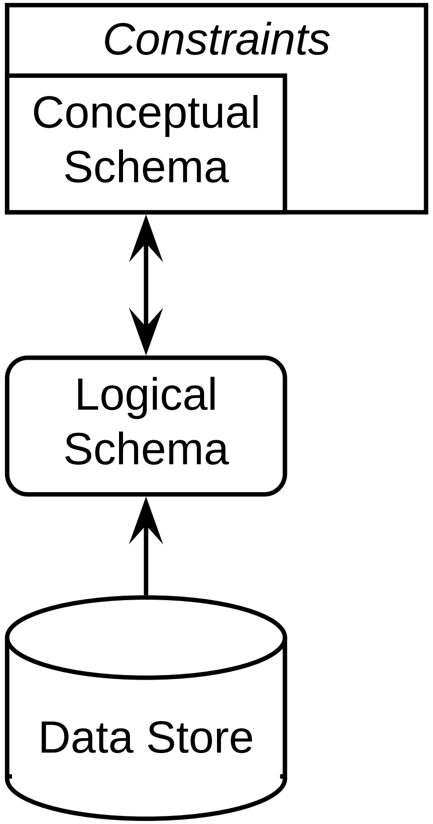
Summary

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

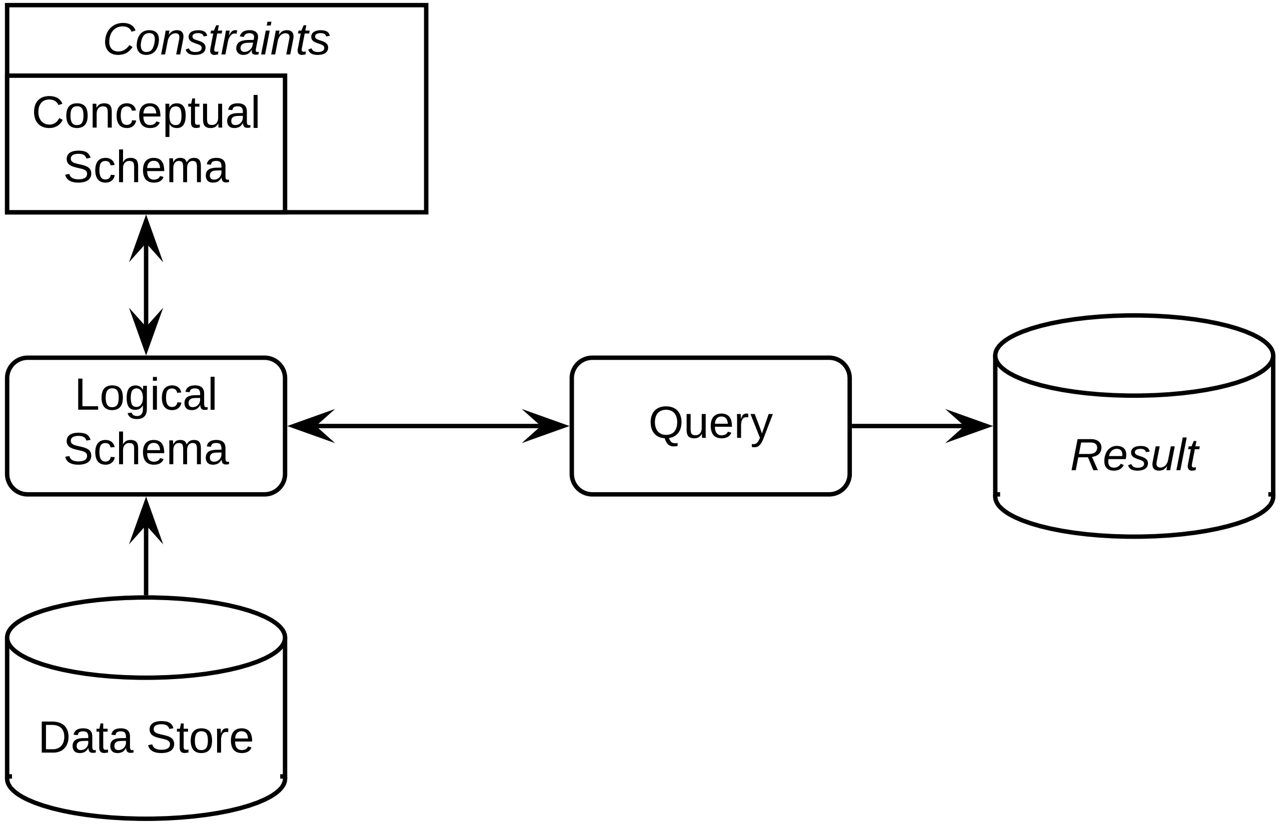
The role of a Conceptual Schema – revisited



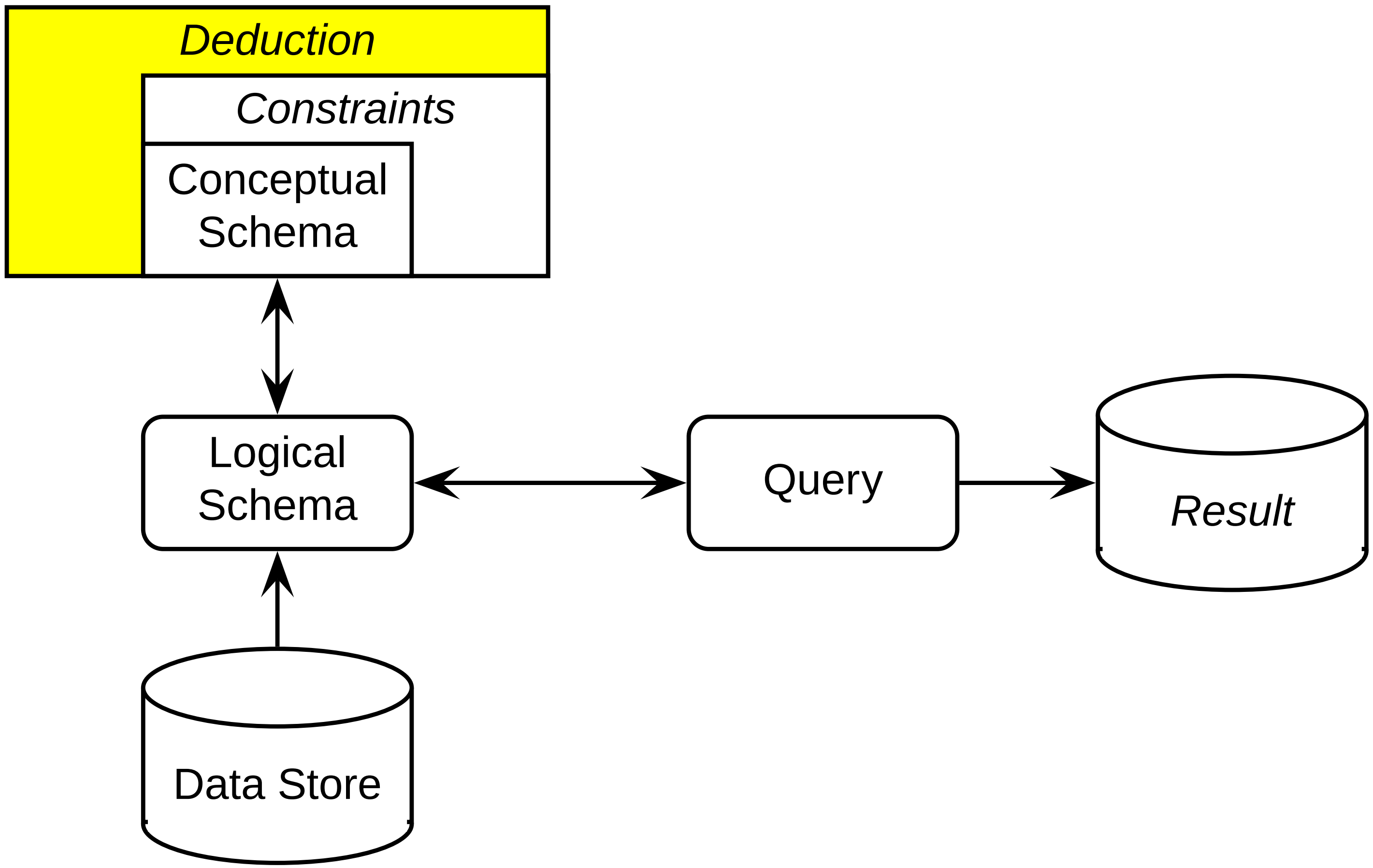
The role of a Conceptual Schema – revisited



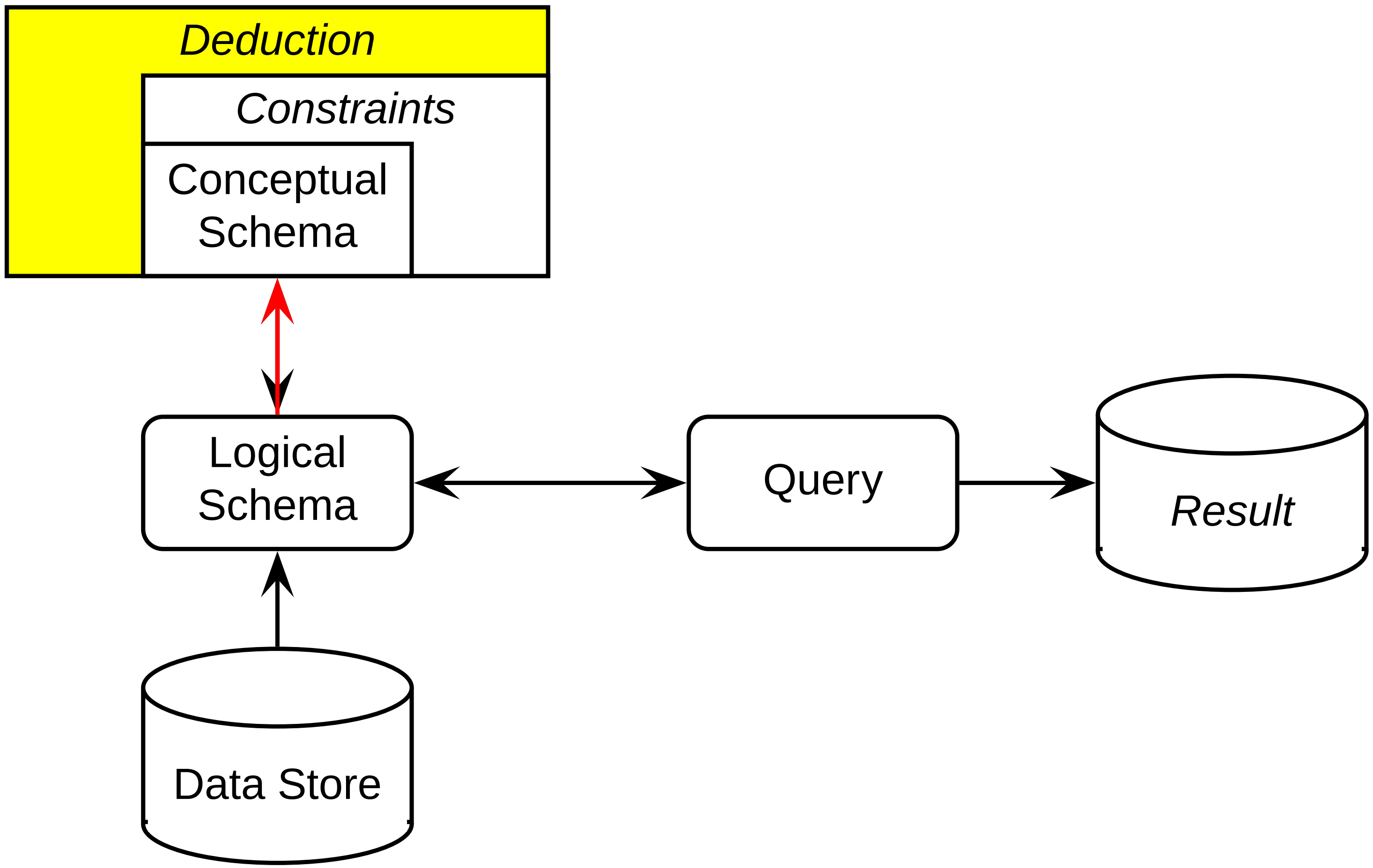
The role of a Conceptual Schema – revisited



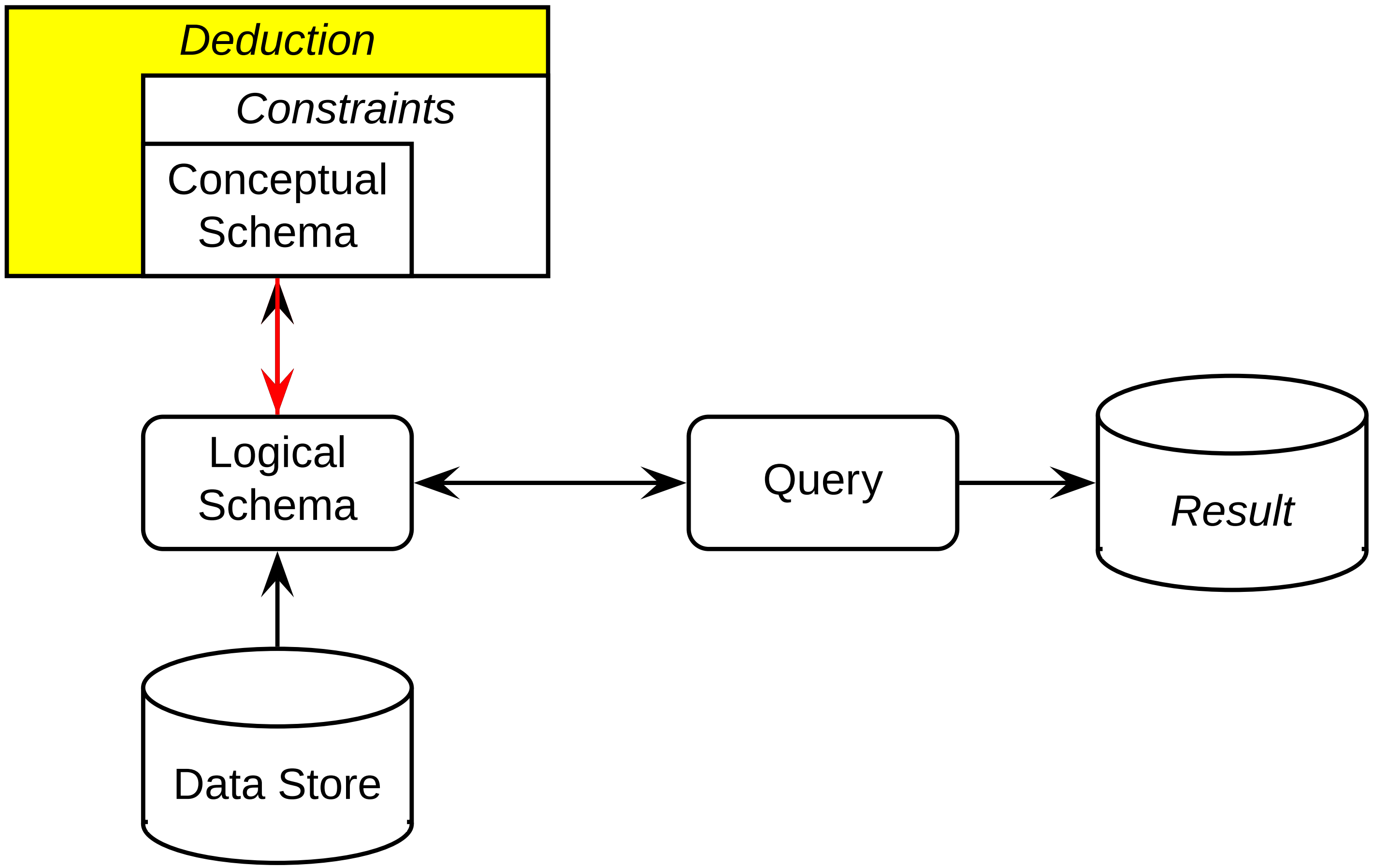
The role of a Conceptual Schema – revisited



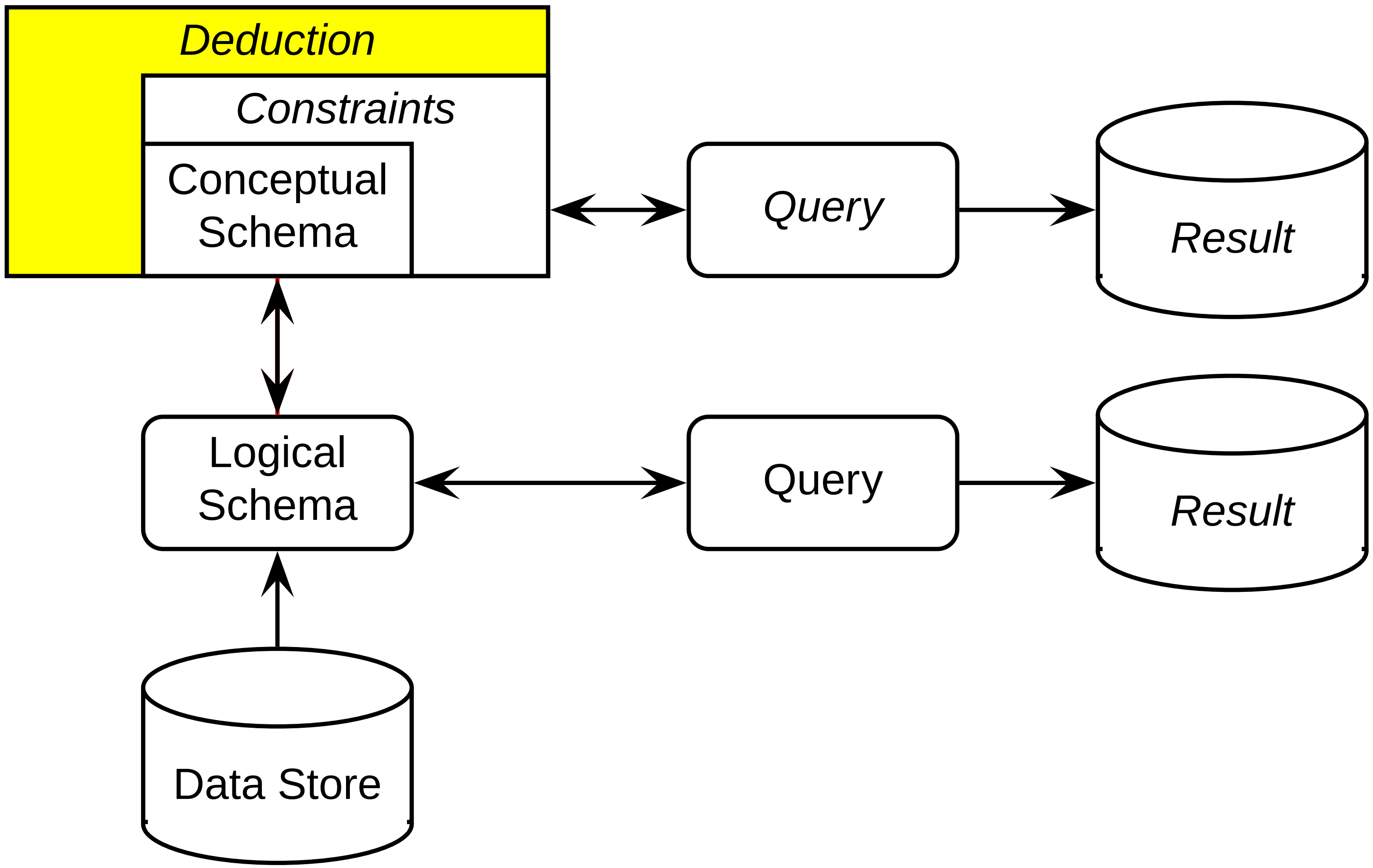
The role of a Conceptual Schema – revisited



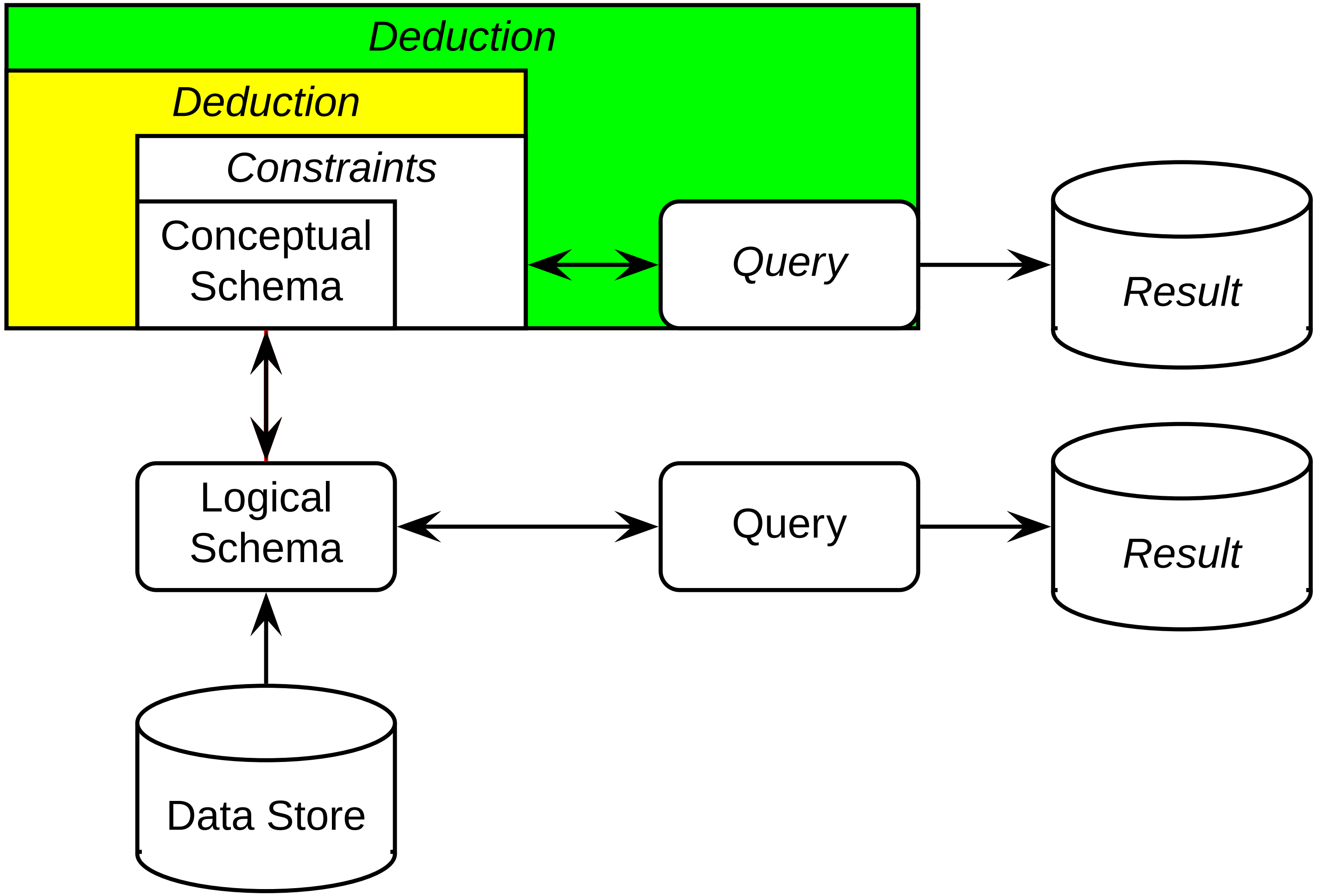
The role of a Conceptual Schema – revisited



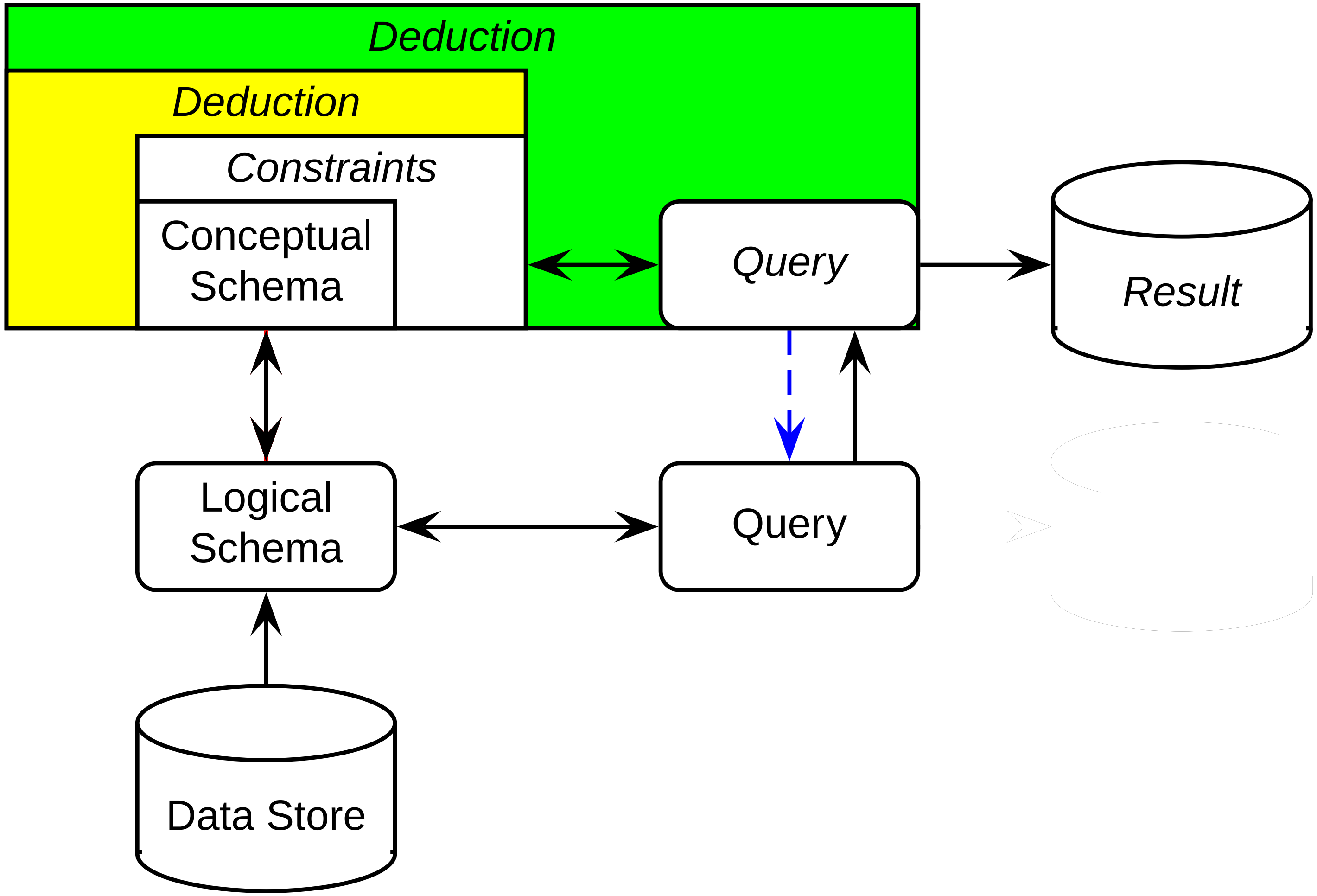
The role of a Conceptual Schema – revisited



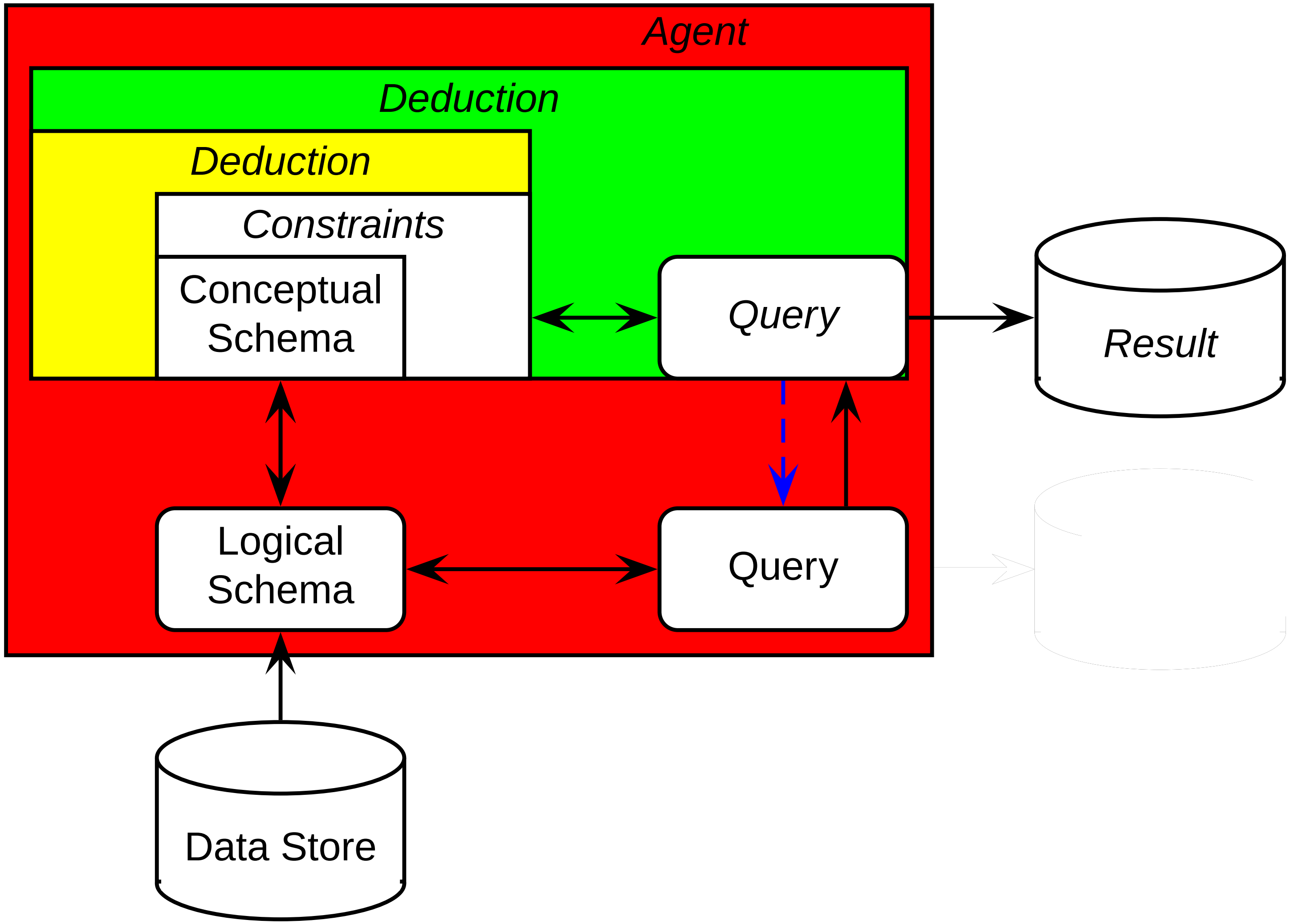
The role of a Conceptual Schema – revisited



The role of a Conceptual Schema – revisited



The role of a Conceptual Schema – revisited



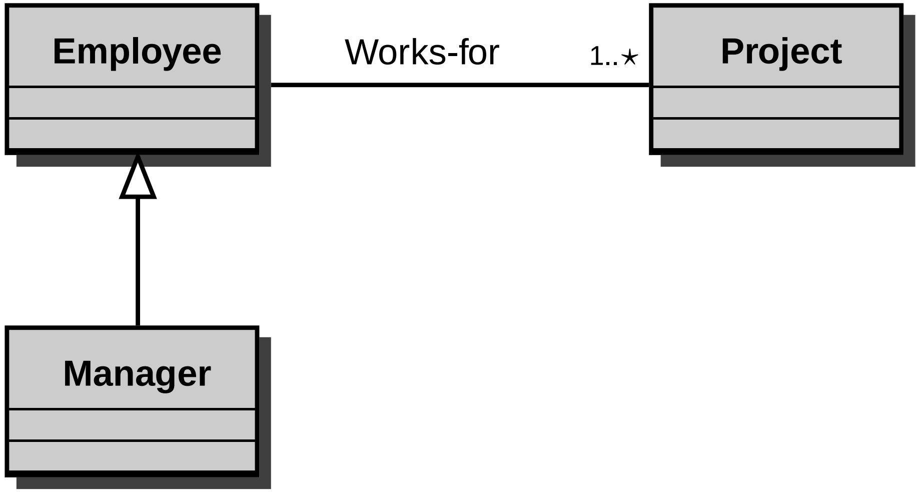
Queries with Ontologies: the DB assumption

- 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

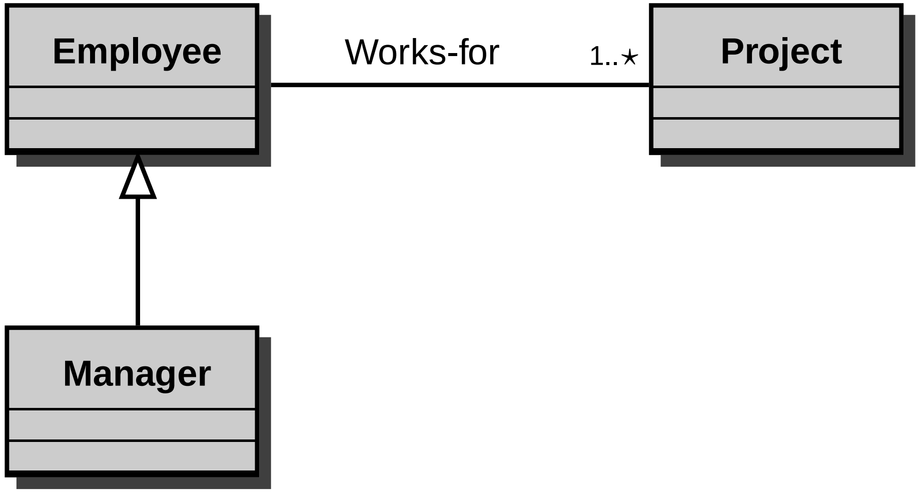
Queries with Ontologies: the DB assumption

- 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



Example



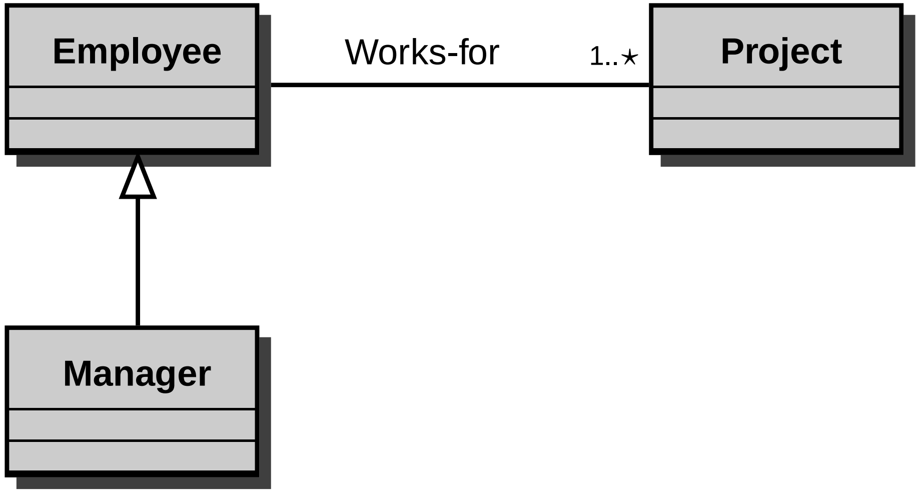
`Employee = { John, Mary, Paul }`

`Manager = { John, Paul }`

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

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

Example




```
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 }

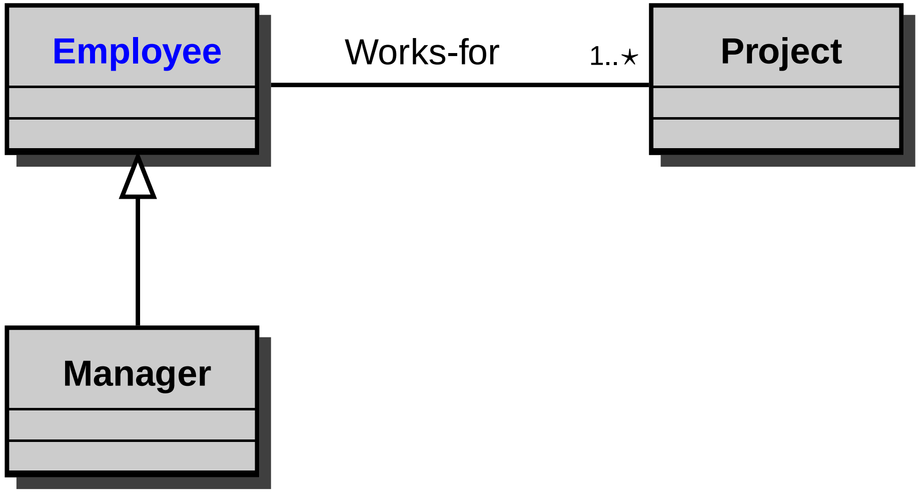
Weakening the DB assumption

- The DB assumption is against the principle that 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

Simple Example

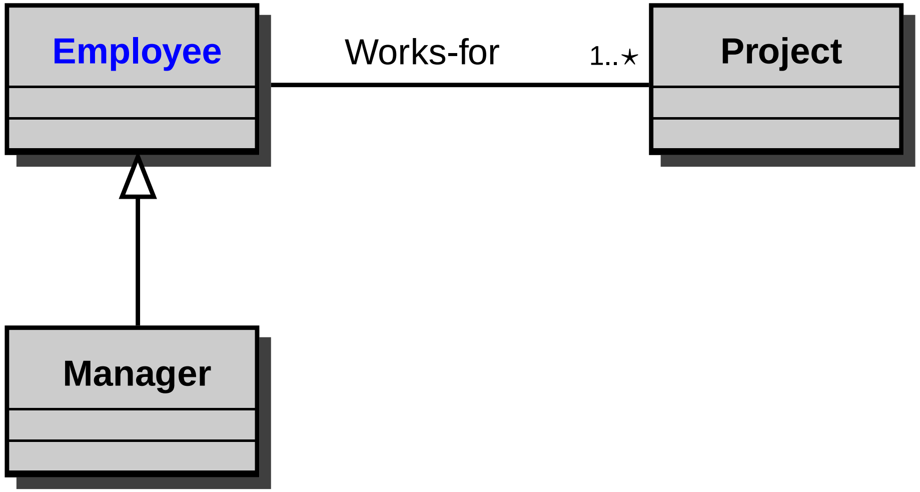


Manager = { **John**, **Paul** }

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

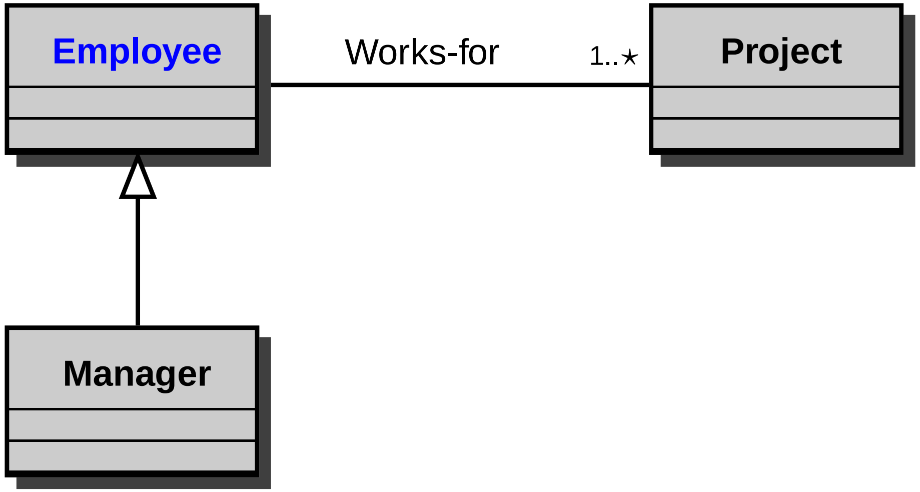
Project = { **Prj-A**, **Prj-B** }

Simple Example



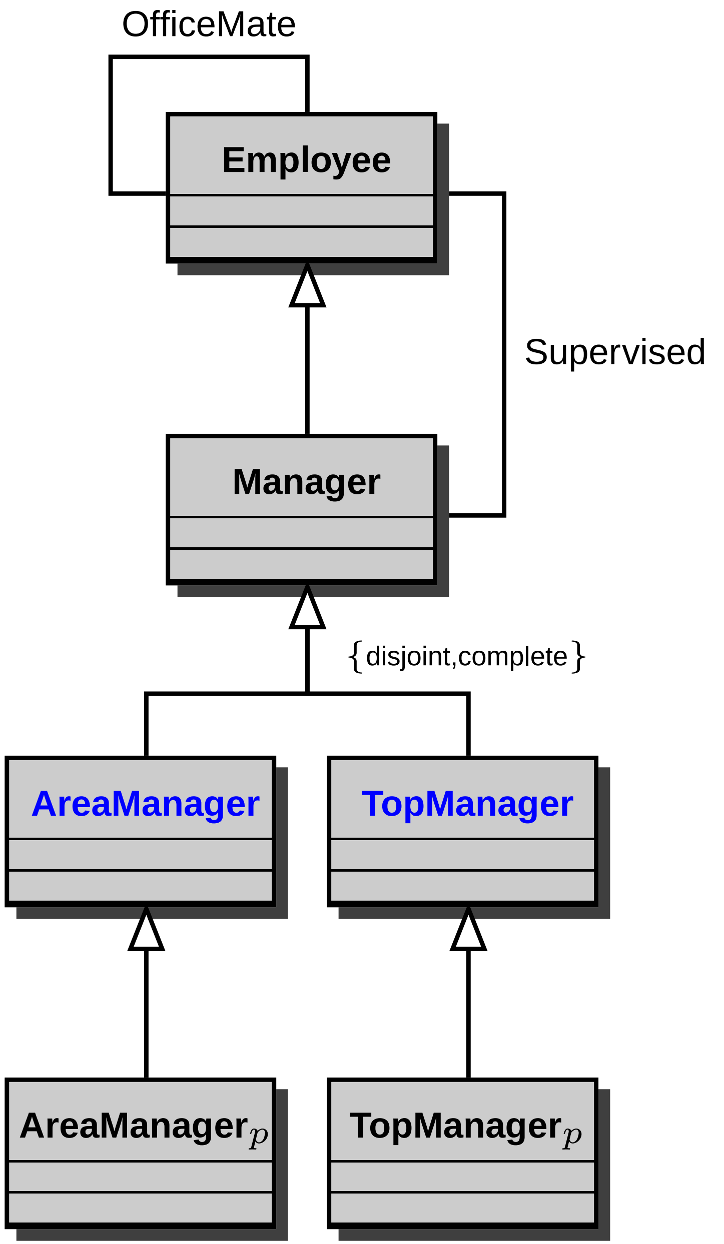

```
Manager = { John, Paul }

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

Project = { Prj-A, Prj-B }

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

Simple Example

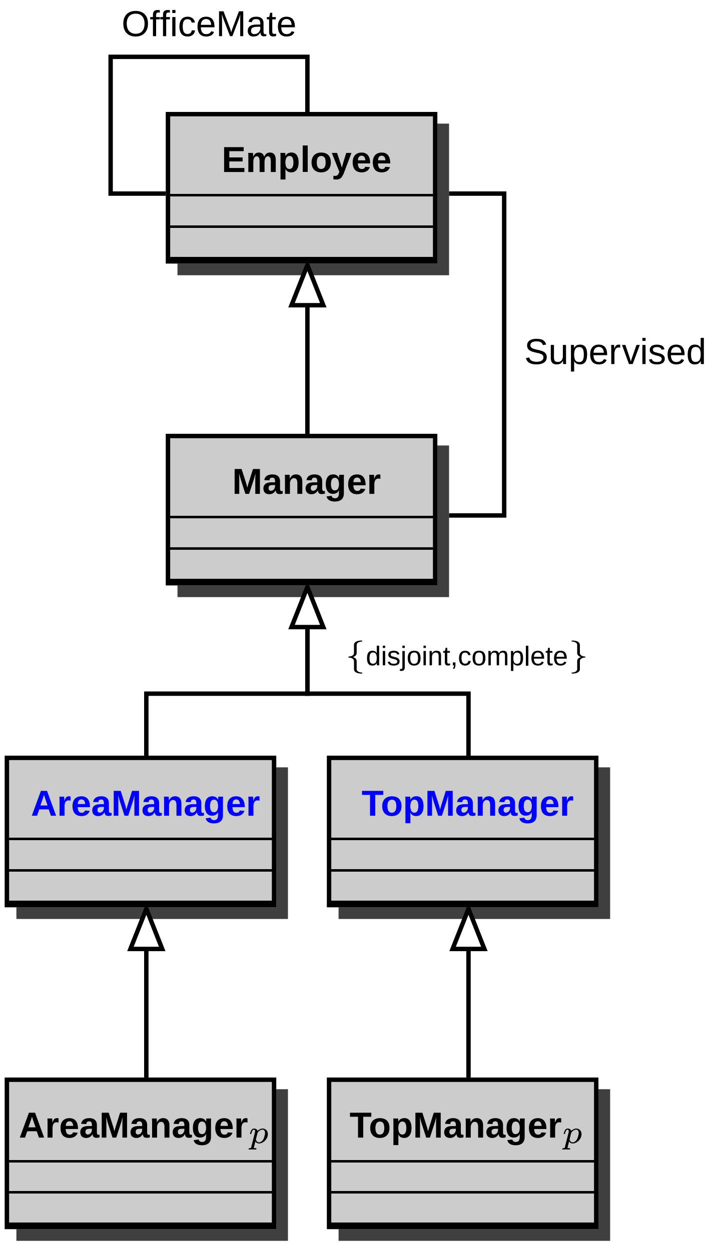



```
Manager = { John, Paul }

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

Project = { Prj-A, Prj-B }

Q(X) :- Employee(X)

⟹ { John, Paul, Mary }
```

Andrea’s Example

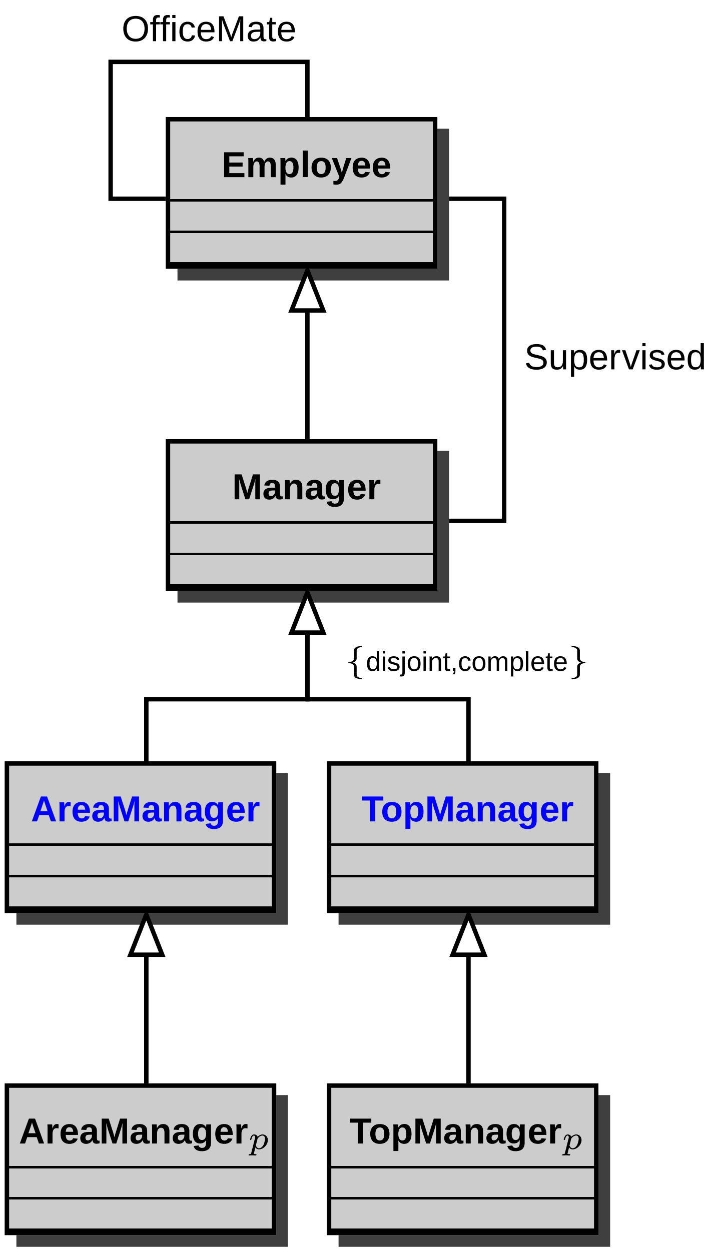


Andrea's Example

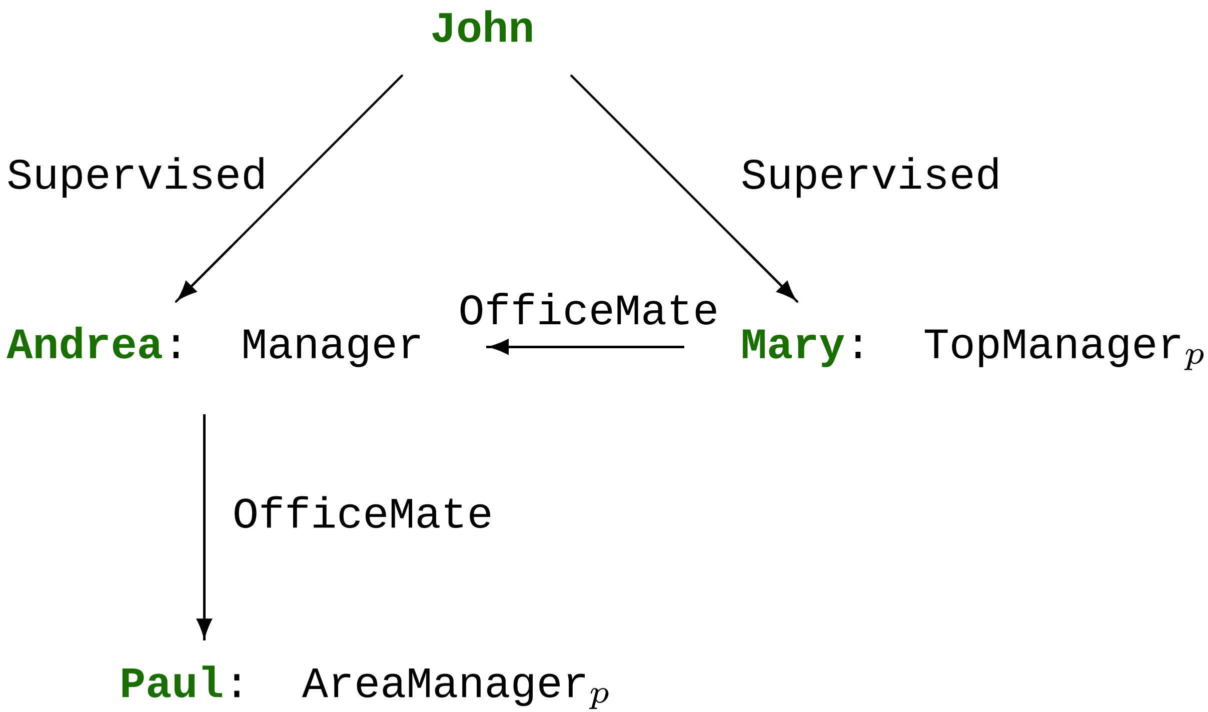


Employee = { **Andrea**, **Paul**, **Mary**, **John** }
Manager = { **Andrea**, **Paul**, **Mary**}
AreaManager$_p$ = { **Paul** }
TopManager$_p$ = { **Mary** }
Supervised = { (**John**,**Andrea**), (**John**,**Mary**) }
OfficeMate = { (**Mary**,**Andrea**), (**Andrea**,**Paul**) }

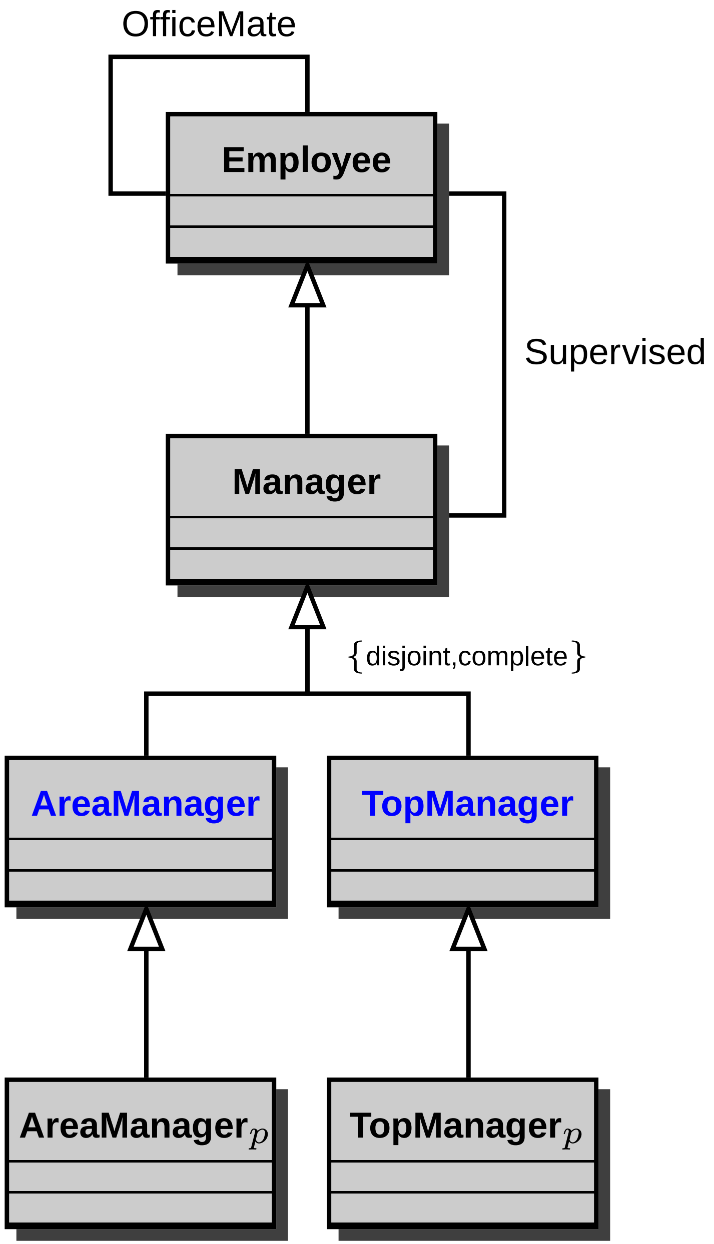
Andrea's Example



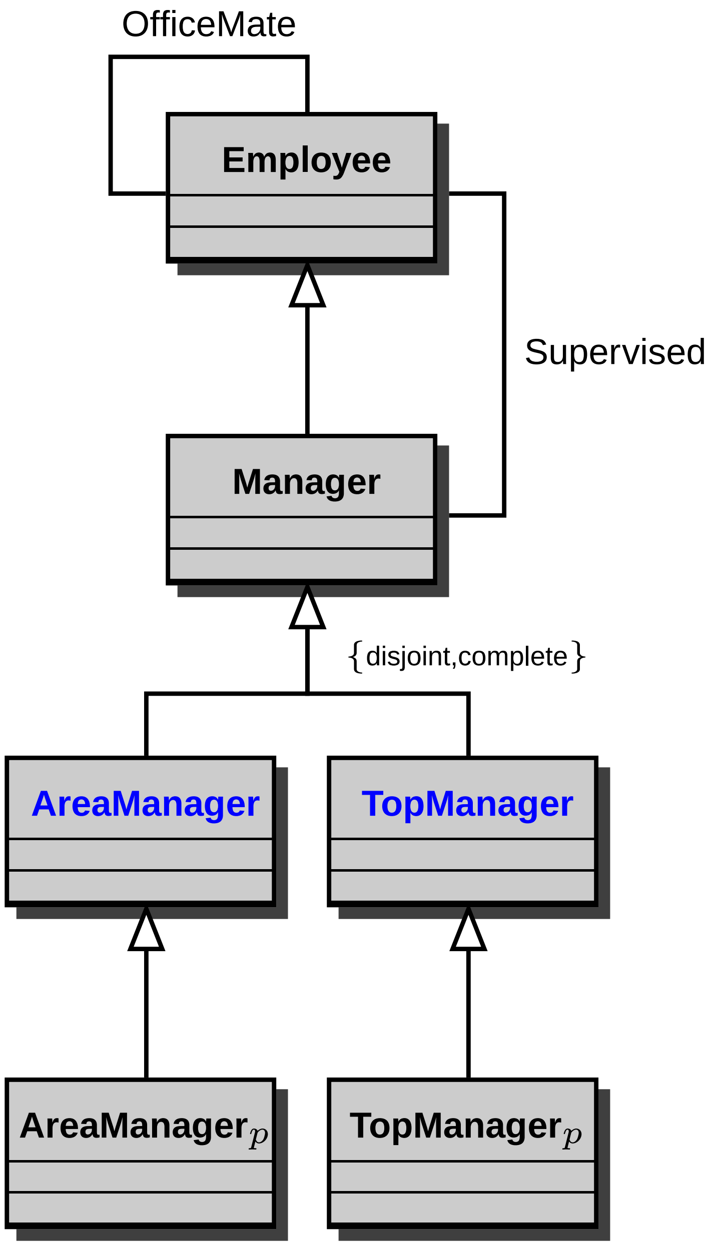
Employee = { **Andrea**, **Paul**, **Mary**, **John** }
Manager = { **Andrea**, **Paul**, **Mary**}
$AreaManager_p$ = { **Paul** }
$TopManager_p$ = { **Mary** }
Supervised = { (**John**,**Andrea**), (**John**,**Mary**) }
OfficeMate = { (**Mary**,**Andrea**), (**Andrea**,**Paul**) }

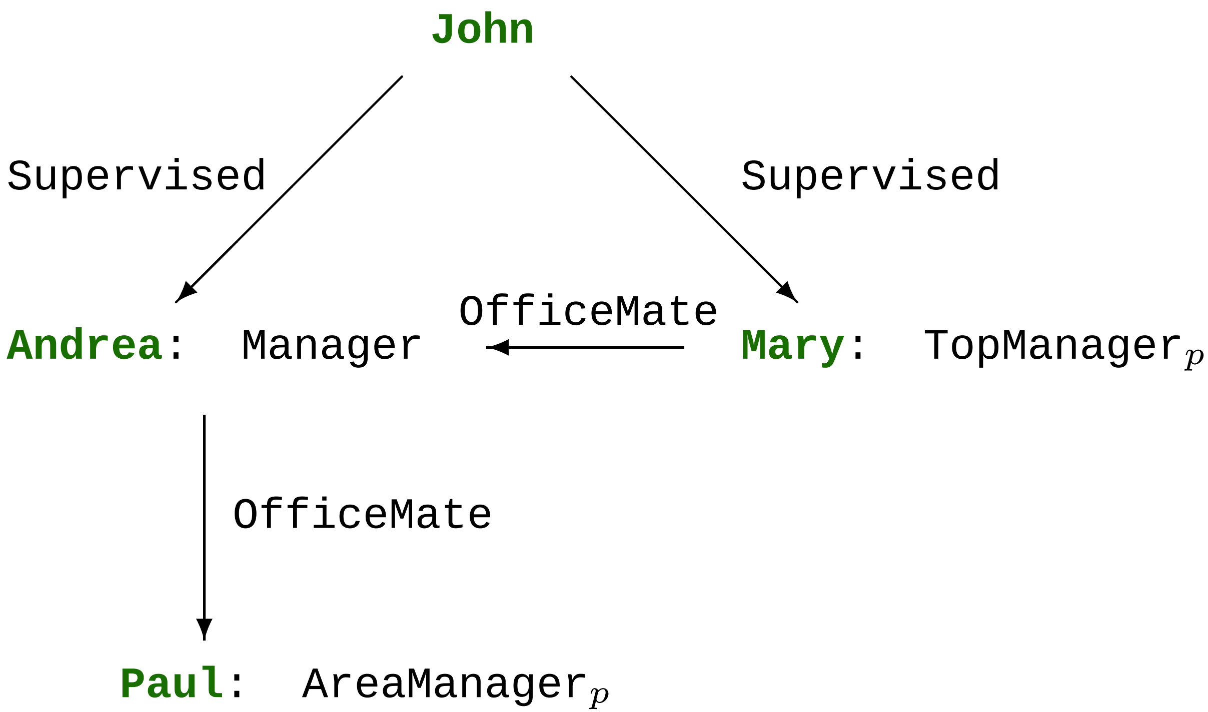


Andrea's Example (cont.)

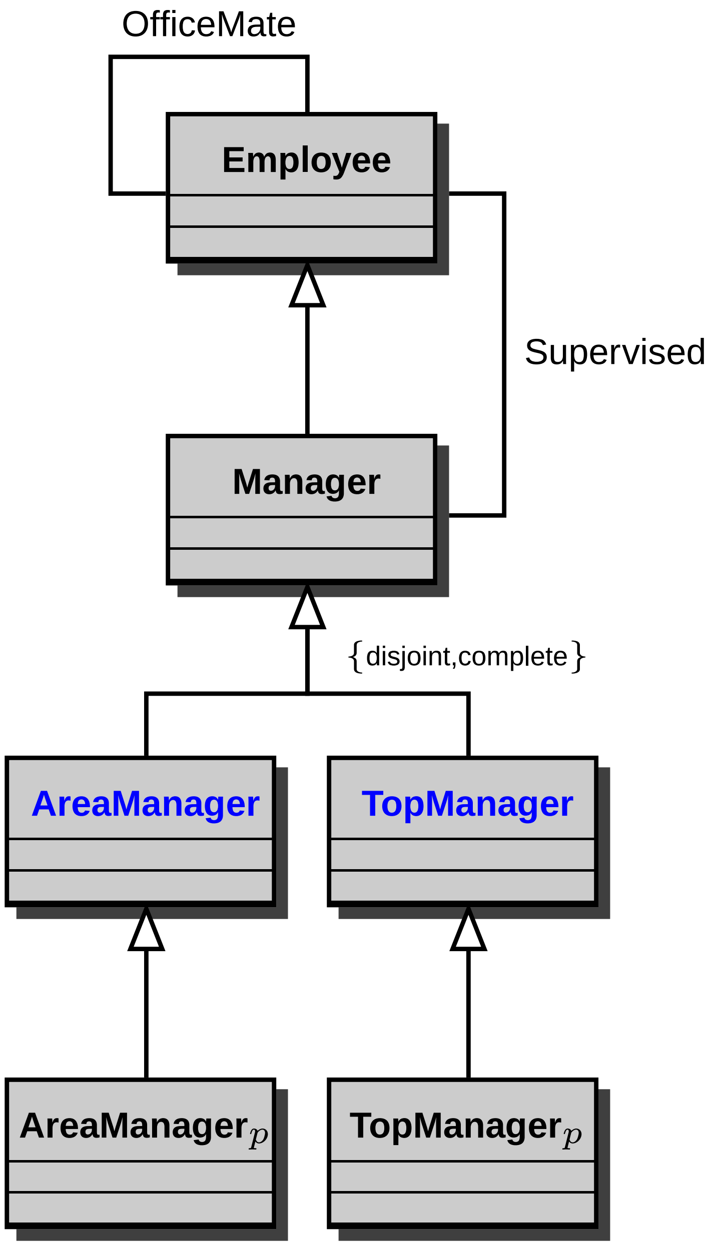


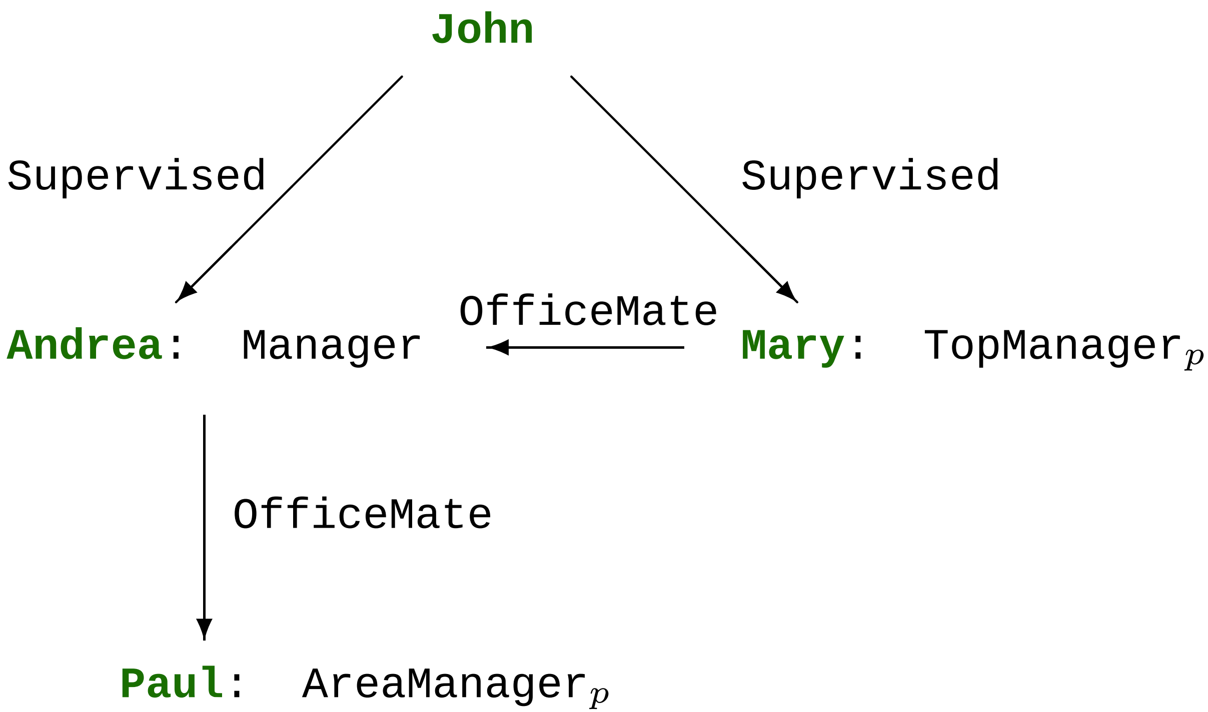
Andrea's Example (cont.)





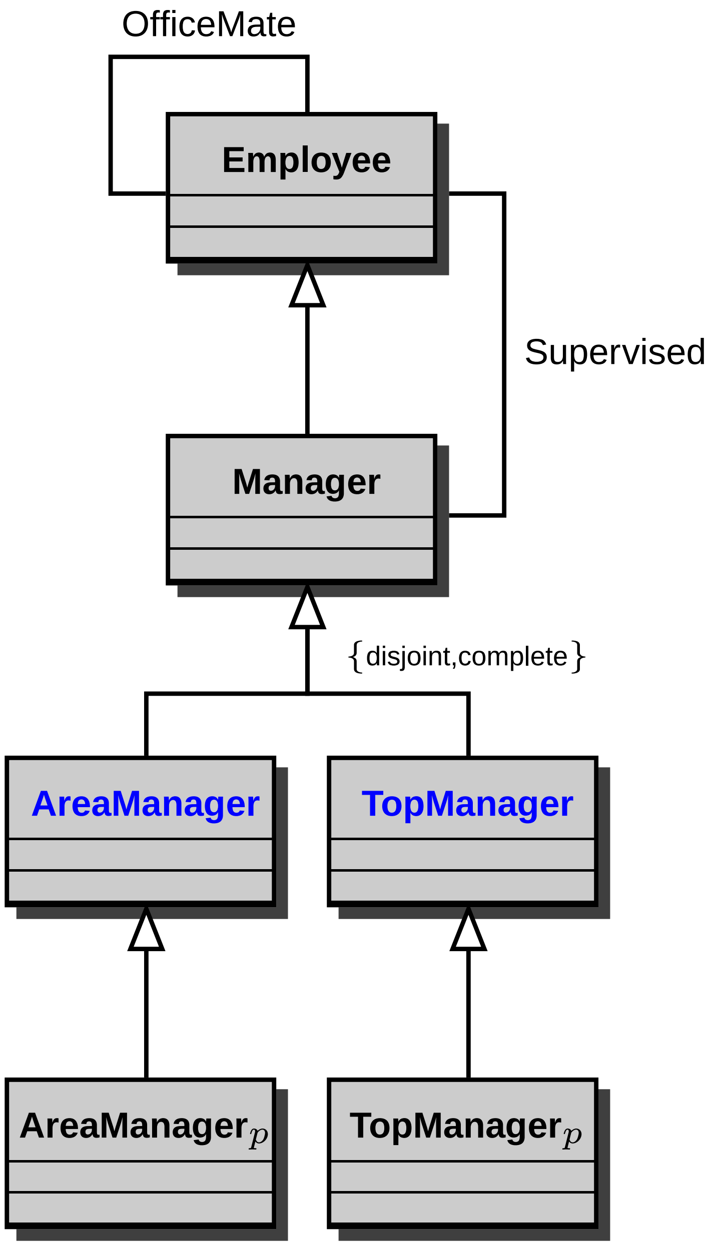
Andrea's Example (cont.)

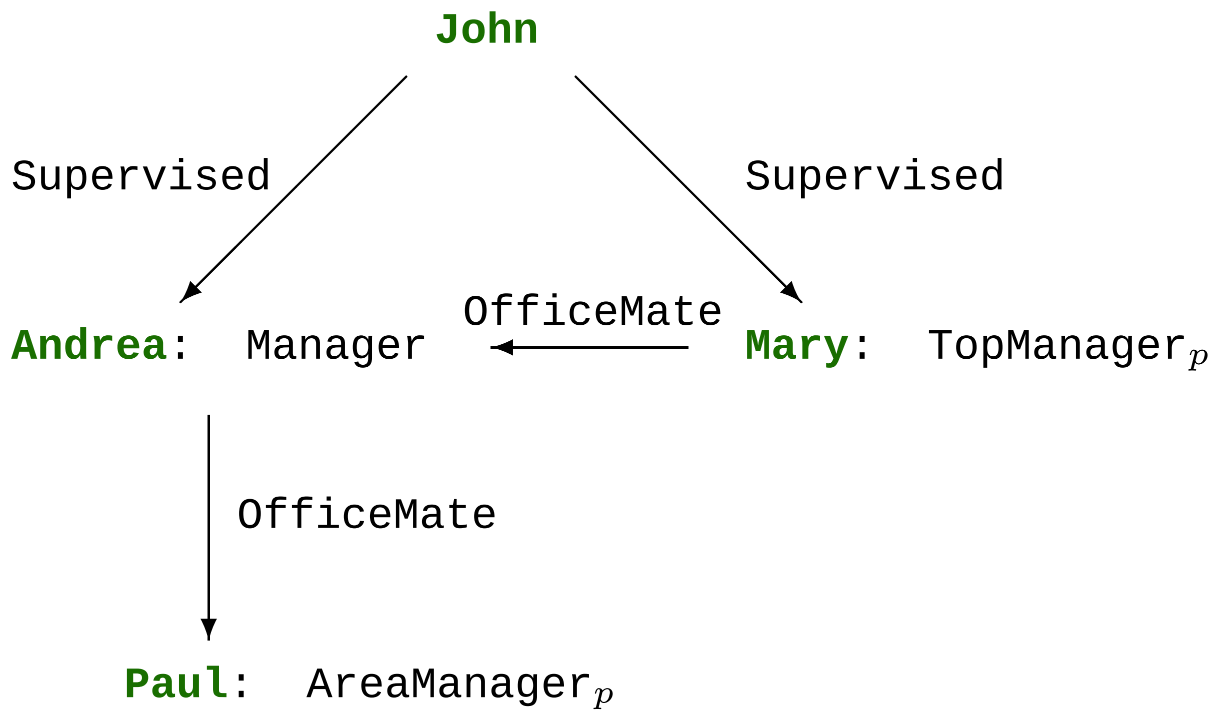





```
Q(X) :- Supervised(X,Y), TopManager(Y),
        Officemate(Y,Z), AreaManager(Z)
```

Andrea's Example (cont.)






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

$\Longrightarrow$ { **John** }

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 views:

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 views:

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

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 views:

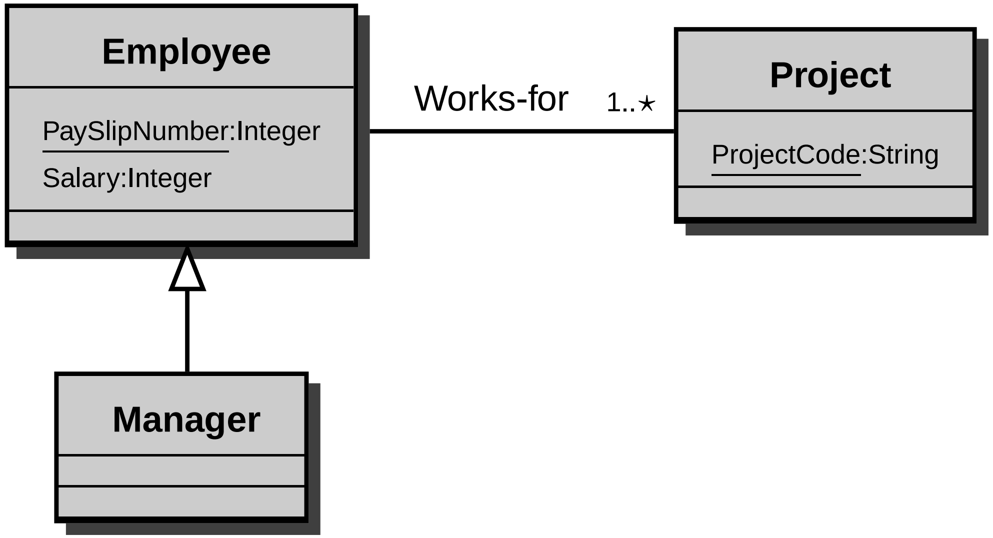
- **GAV** (global-as-view): a view over the information source is given for *some* term in the ontology;
  - an ER schema can be easily mapped to its corresponding relational schema in normal form via a total GAV mapping.

View based query processing

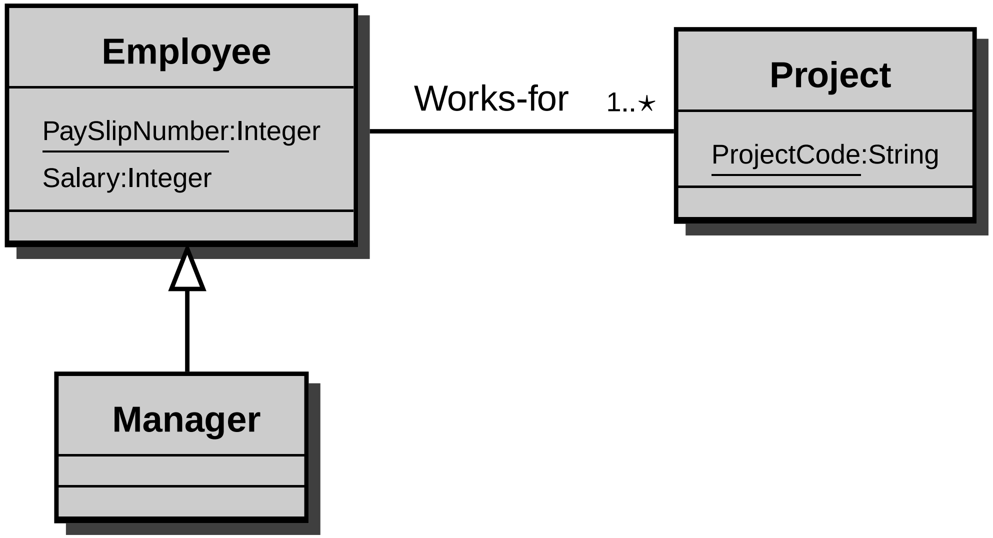
In general, the **mapping** between the ontology and the information source terms can be given in terms of a set of views:

- **GAV** (global-as-view): a view over the information source is given for *some* term in the ontology;
  - an ER schema can be easily mapped to its corresponding relational schema in normal form via a total 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.

Total GAV mapping



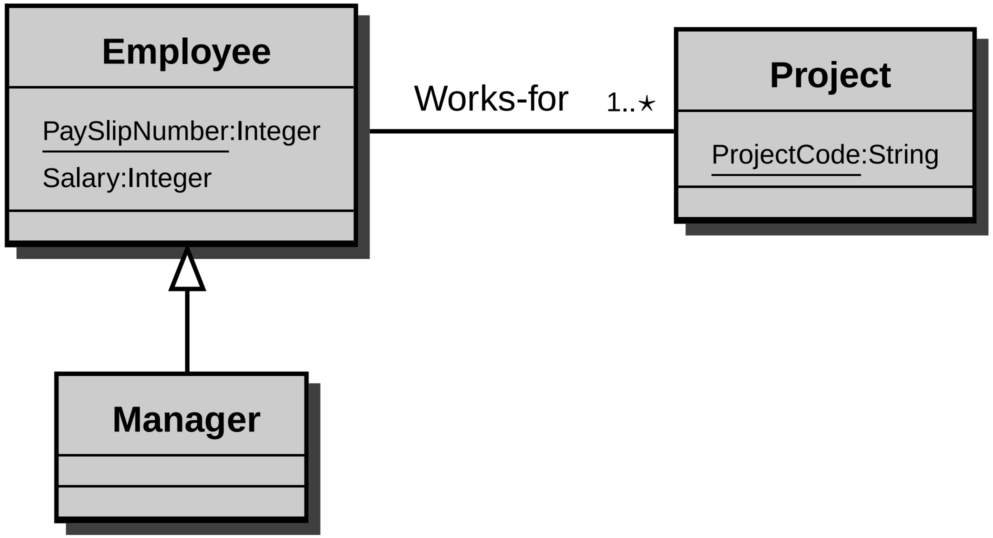
Total GAV mapping



`NF-Employee(PaySlipNumber,Salary,ManagerP)`

`NF-Works-for(PaySlipNumber,ProjectCode)`

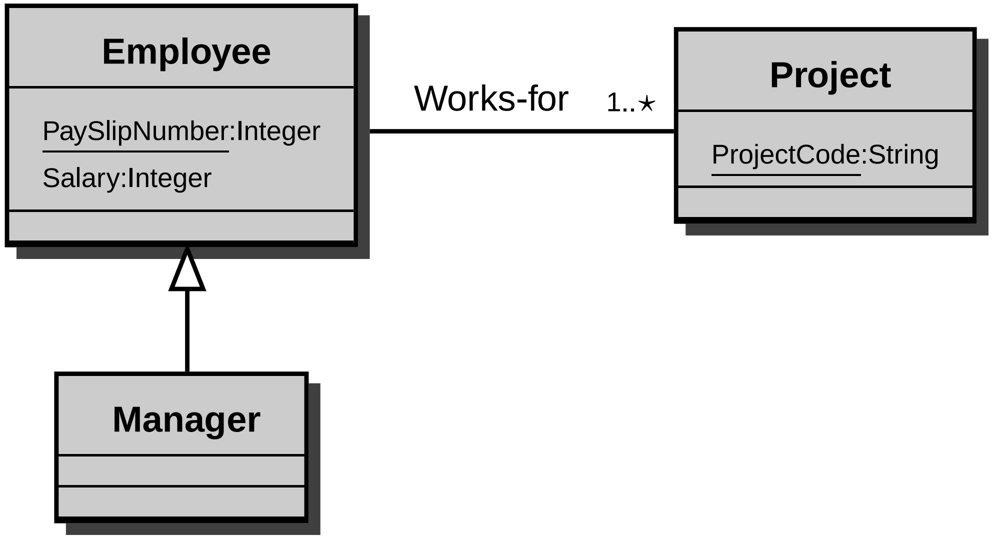
Total GAV mapping



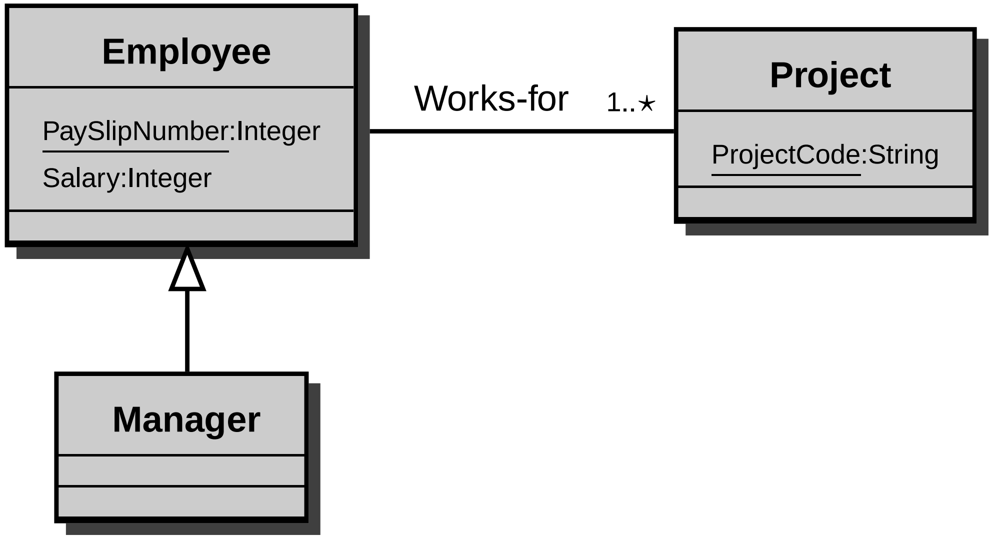

```
NF-Employee(PaySlipNumber,Salary,ManagerP)

NF-Works-for(PaySlipNumber,ProjectCode)
```

```
Employee(X) :- NF-Employee(X,Y,Z)

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

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

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

Project(X) :- NF-Works-for(X,Y)
```

LAV mapping


Employee
PaySlipNumber:Integer
Salary:Integer
Works-for
1..⋆
Project
ProjectCode:String
Manager

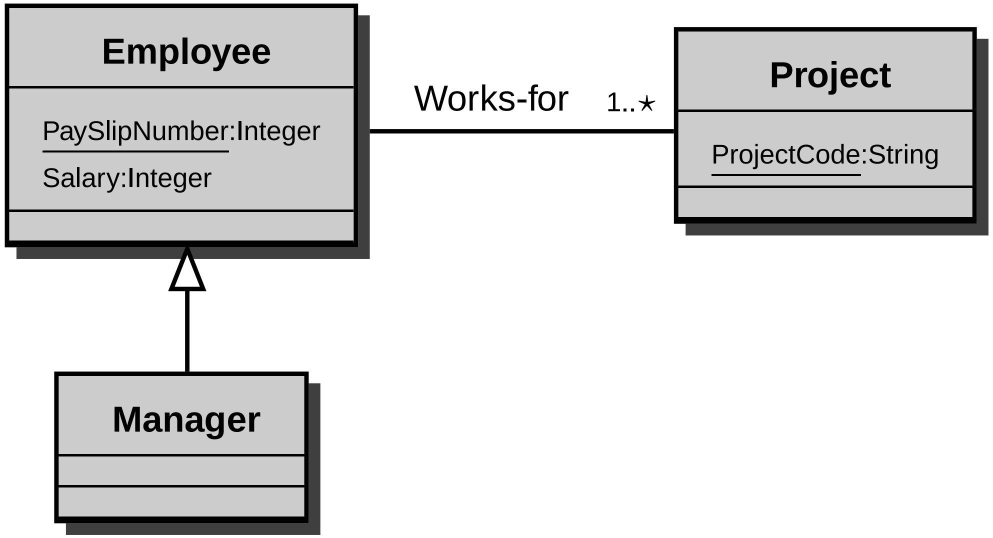
LAV mapping



NF-Employee(PaySlipNumber,Salary,ManagerP)

NF-Works-for(PaySlipNumber,ProjectCode)

LAV mapping



NF-Employee(PaySlipNumber,Salary,ManagerP)

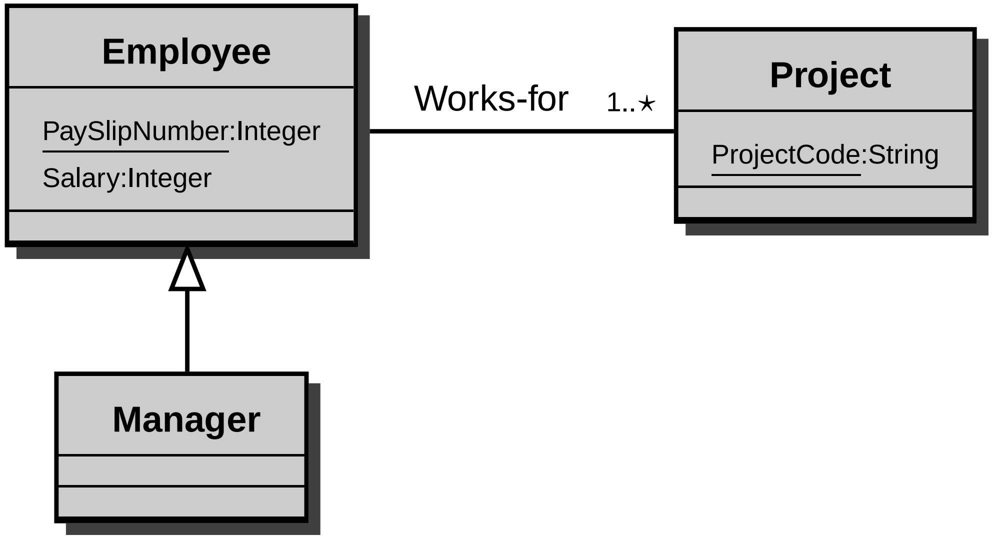
NF-Works-for(PaySlipNumber,ProjectCode)

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

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

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

Queries with LAV mapping



NF-Employee(PaySlipNumber,Salary,ManagerP)

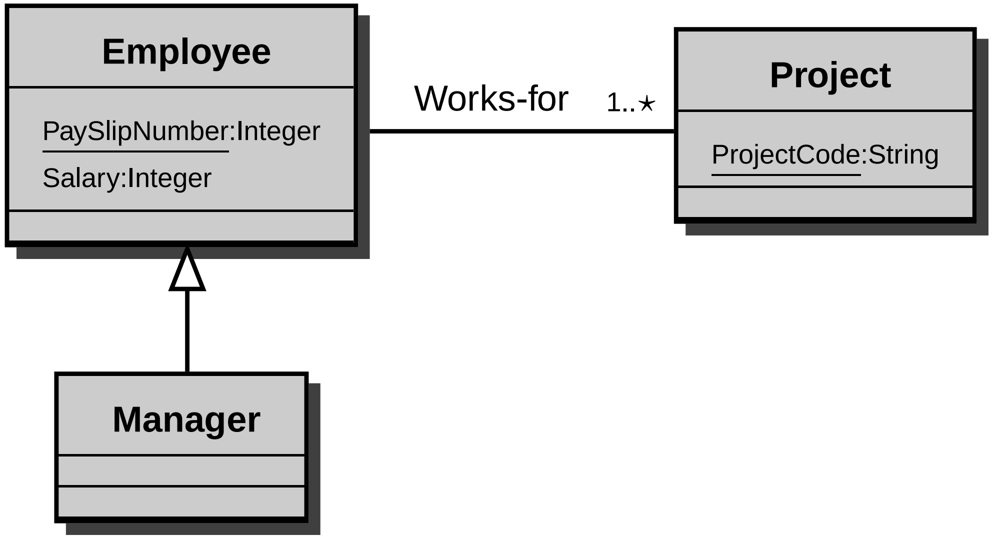
NF-Works-for(PaySlipNumber,ProjectCode)

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

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

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

Queries with LAV mapping



NF-Employee(PaySlipNumber,Salary,ManagerP)

NF-Works-for(PaySlipNumber,ProjectCode)

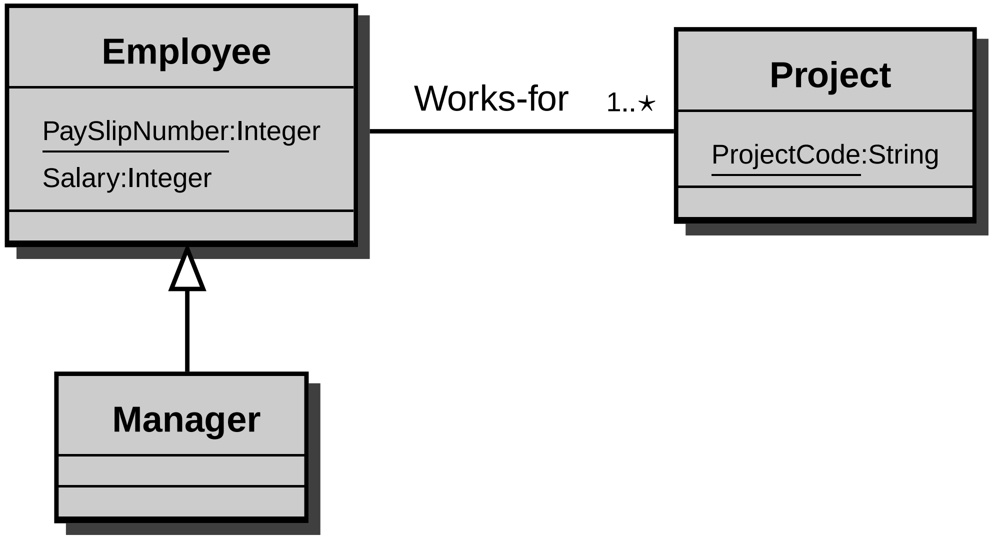
NF-Employee(X,Y,Z) :- Manager(X), Salary(X,Y), Z=true

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

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

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

Queries with LAV mapping



`NF-Employee(PaySlipNumber,Salary,ManagerP)`

`NF-Works-for(PaySlipNumber,ProjectCode)`

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

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

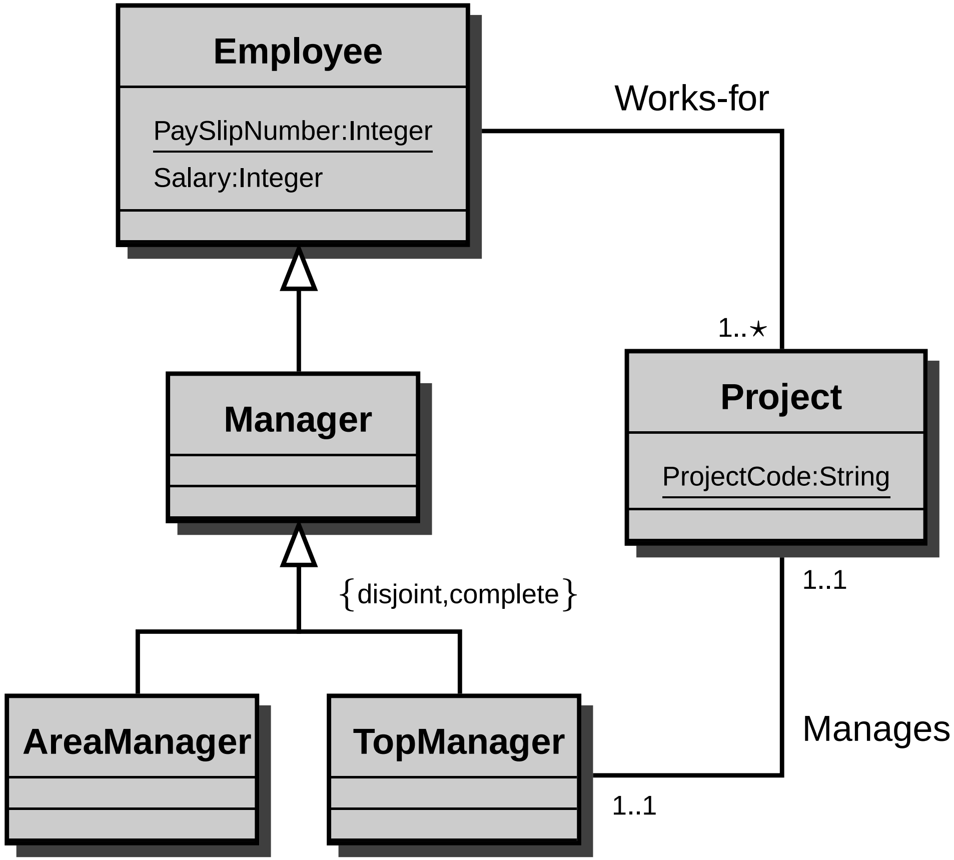
`NF-Works-for(X,Y) :- Works-for(X,Y)`

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

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

Reasoning over queries

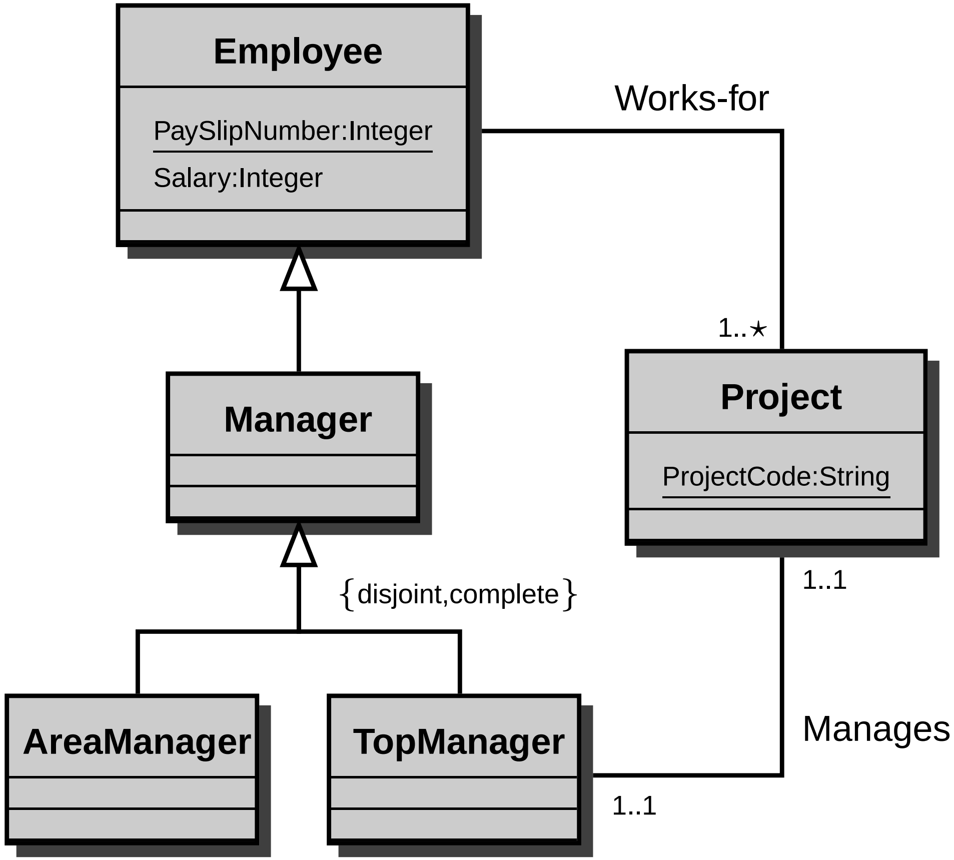
```
Q(X,Y) :- Employee(X), Works-for(X,Y), Manages(X,Y)
```



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

Reasoning over queries

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



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

$\rightsquigarrow$ INCONSISTENT QUERY!

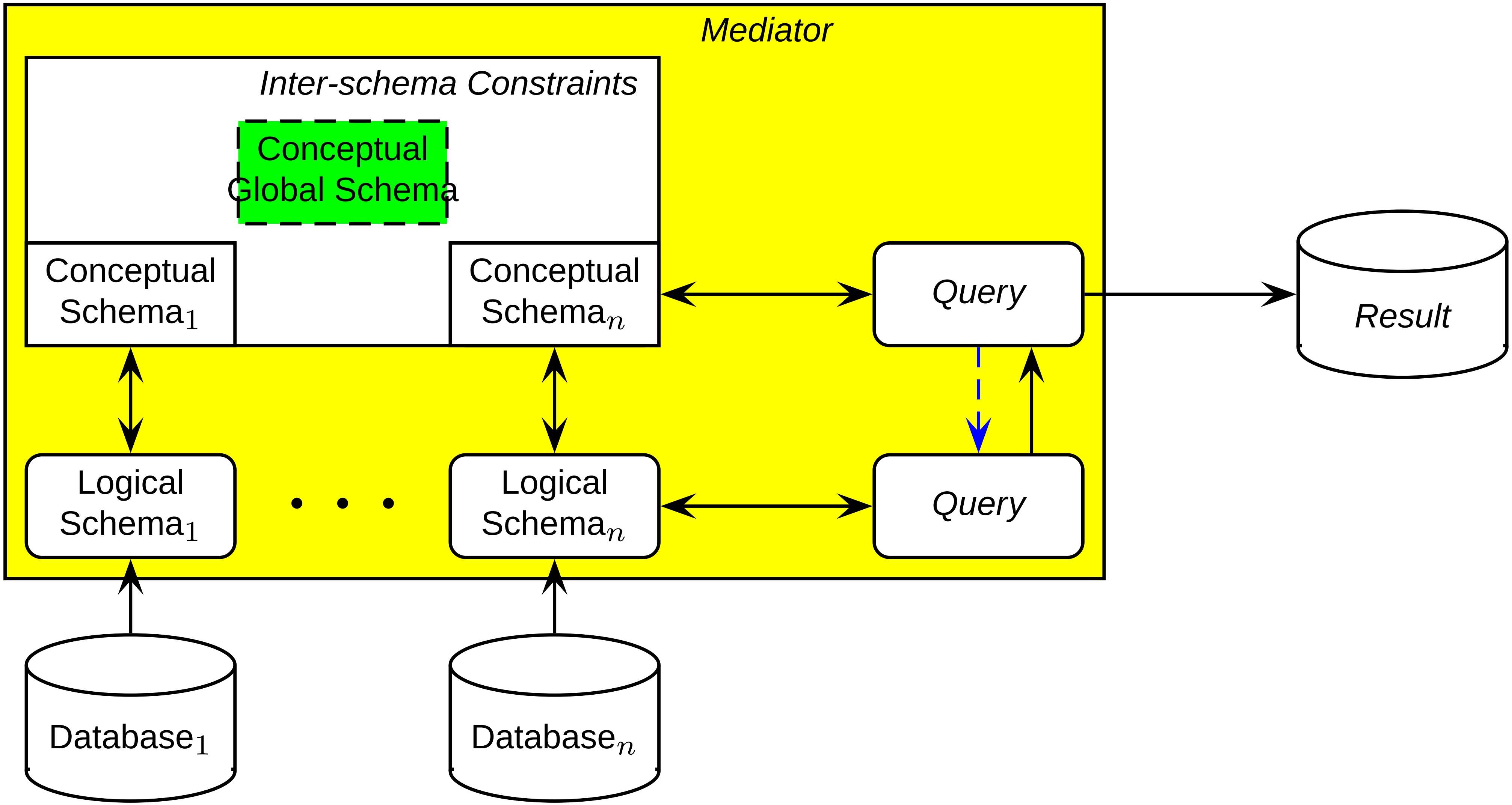
Summary

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

Usefulness of View-based Query Processing

- In data integration, the views represent the only information sources accessible to answer a query.
- A data warehouse can be seen as a set of materialised views, and, therefore, query processing reduces to view-based query answering.
- In query optimisation, view-based query processing is relevant because using the views may speed up query processing.
- Since the views provide partial knowledge on the database, view-based query processing can be seen as a special case query answering with incomplete information.

Mediator Architecture for Ontology Integration



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!