Conceptual schemas and ontologies for data access: myths and challenges

Enrico Franconi

KRDB Research Centre for Knowledge and Data
Free University of Bozen-Bolzano, Italy

http://krdb.eu/franconi

ESAO 2021

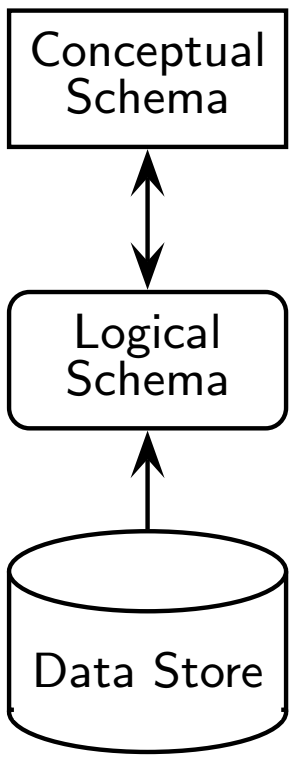
Summary of this talk

- What is a Conceptual Schema
- The role of a Conceptual Schema in Databases
- Accessing Data via a Conceptual Schema
- DBoxes
- ABoxes

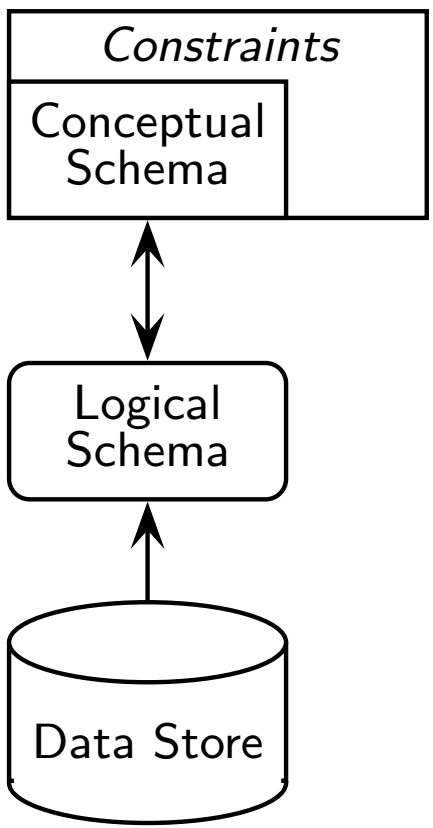
What is a Conceptual Schema

- A semantic layer gives an abstract perspective to a system, different from its original representation.
- Its purpose is to introduce a vocabulary understandable by the agents that need to interact with the system.
- A semantic layer is a formal conceptualisation of the agents' perspective: a conceptual schema (ontology, logical theory).
- A conceptual schema includes a set of constraints (the *semantics*) over the given vocabulary, which specifies what should hold in any possible configuration of the system.
- Any possible configuration of the system (an instance) conforms to the constraints expressed by the conceptual schema.
- Given a conceptual schema, a *legal instance* is a finite possible configuration of the system satisfying the constraints.
- We focus on database systems, their conceptual schemas, and their legal database instances.

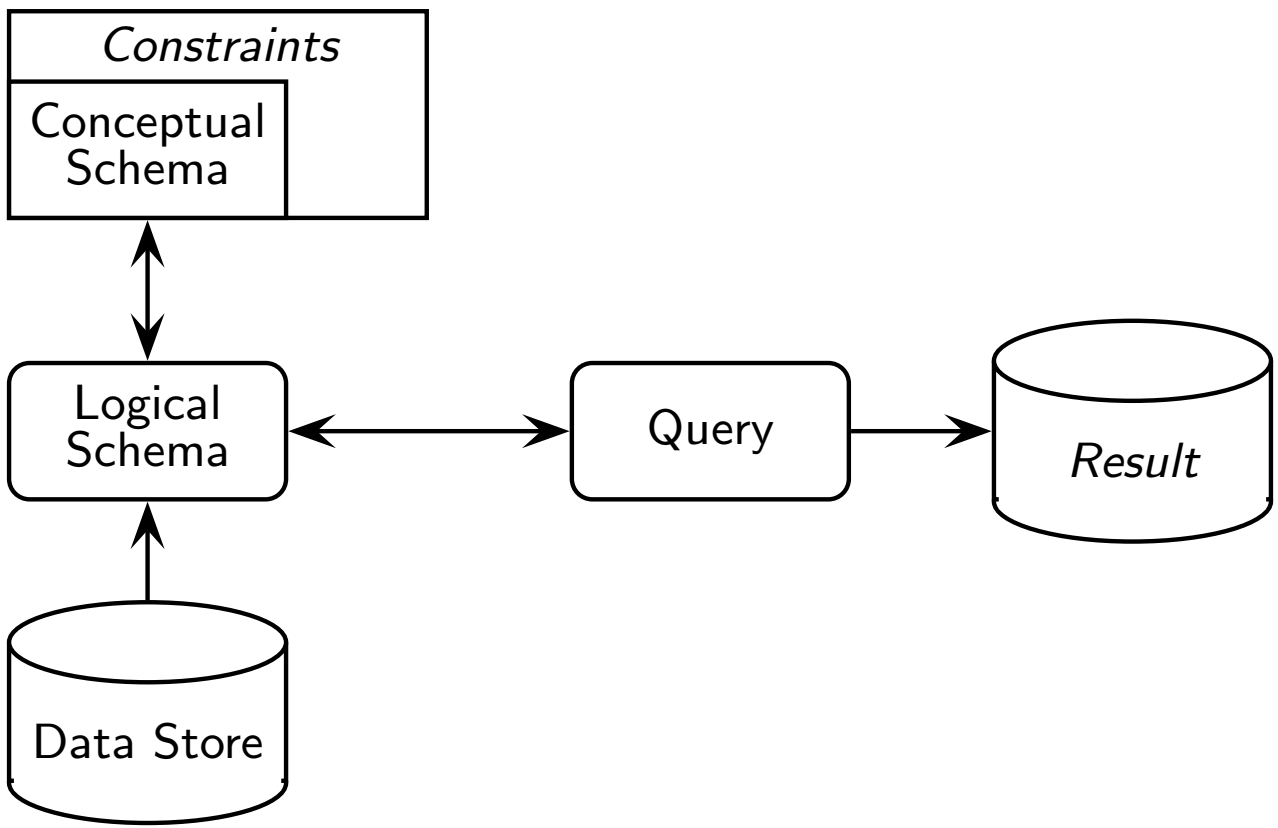
The role of a Conceptual Schema in Databases



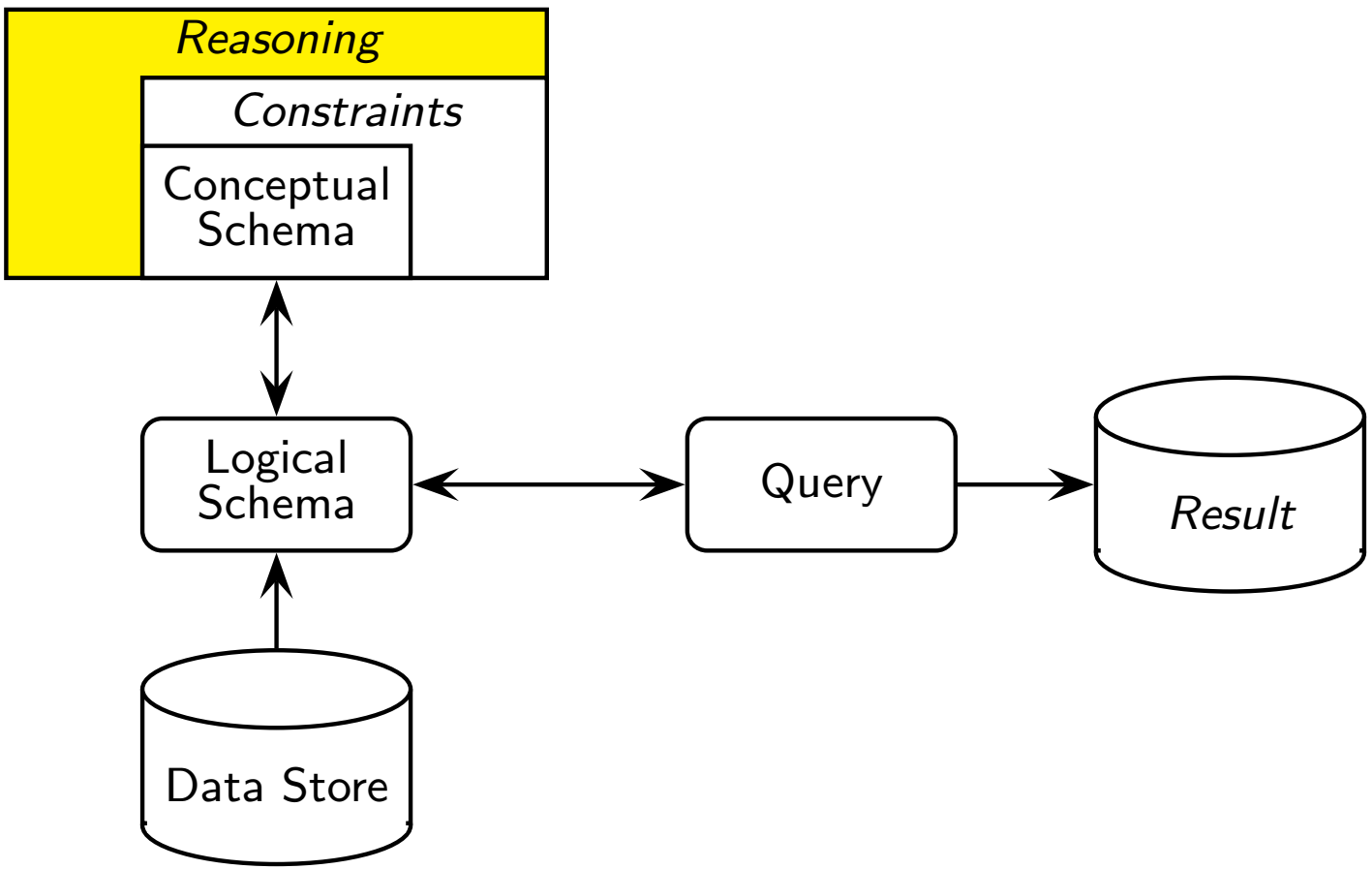
The role of a Conceptual Schema in Databases



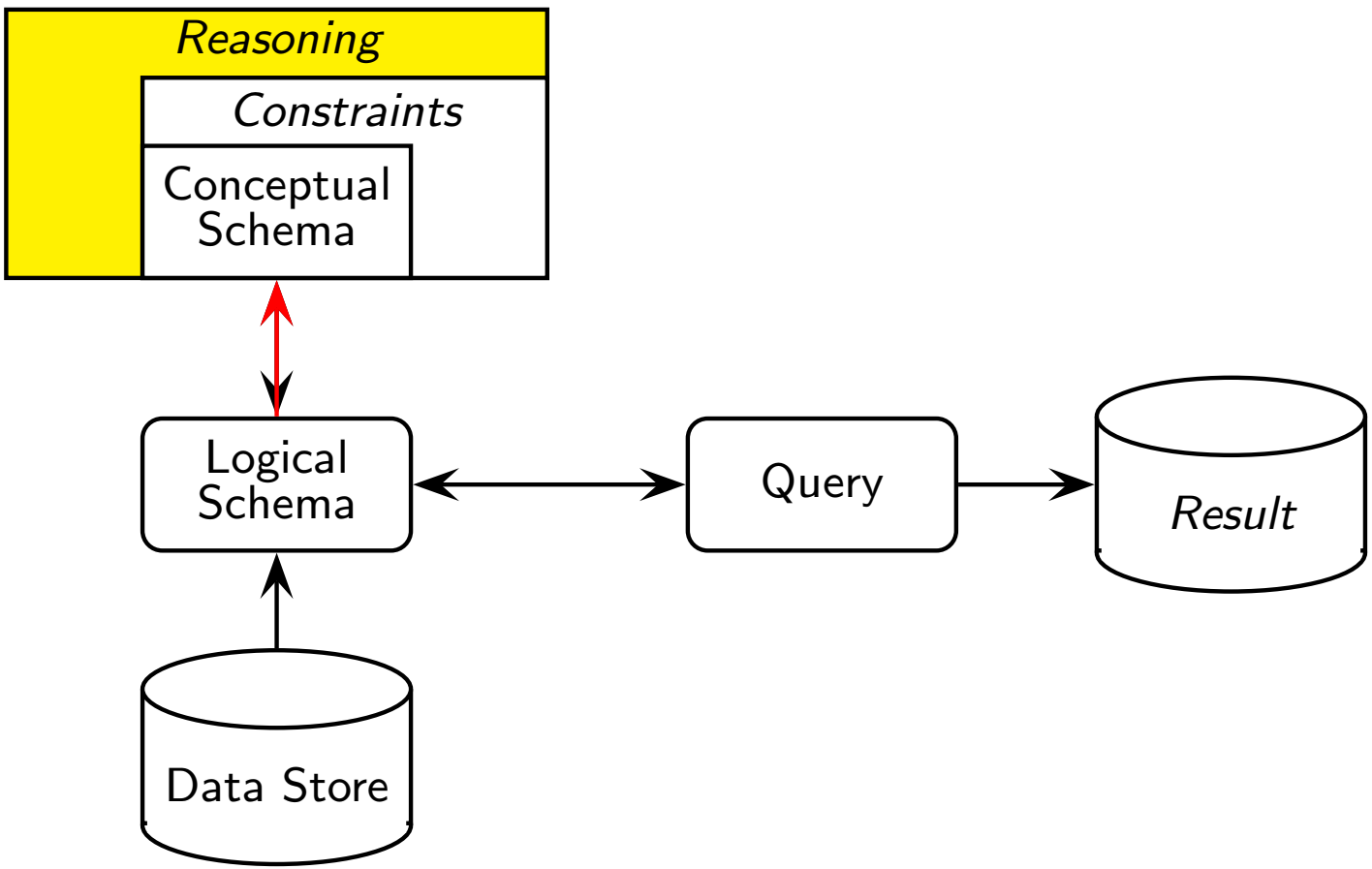
The role of a Conceptual Schema in Databases



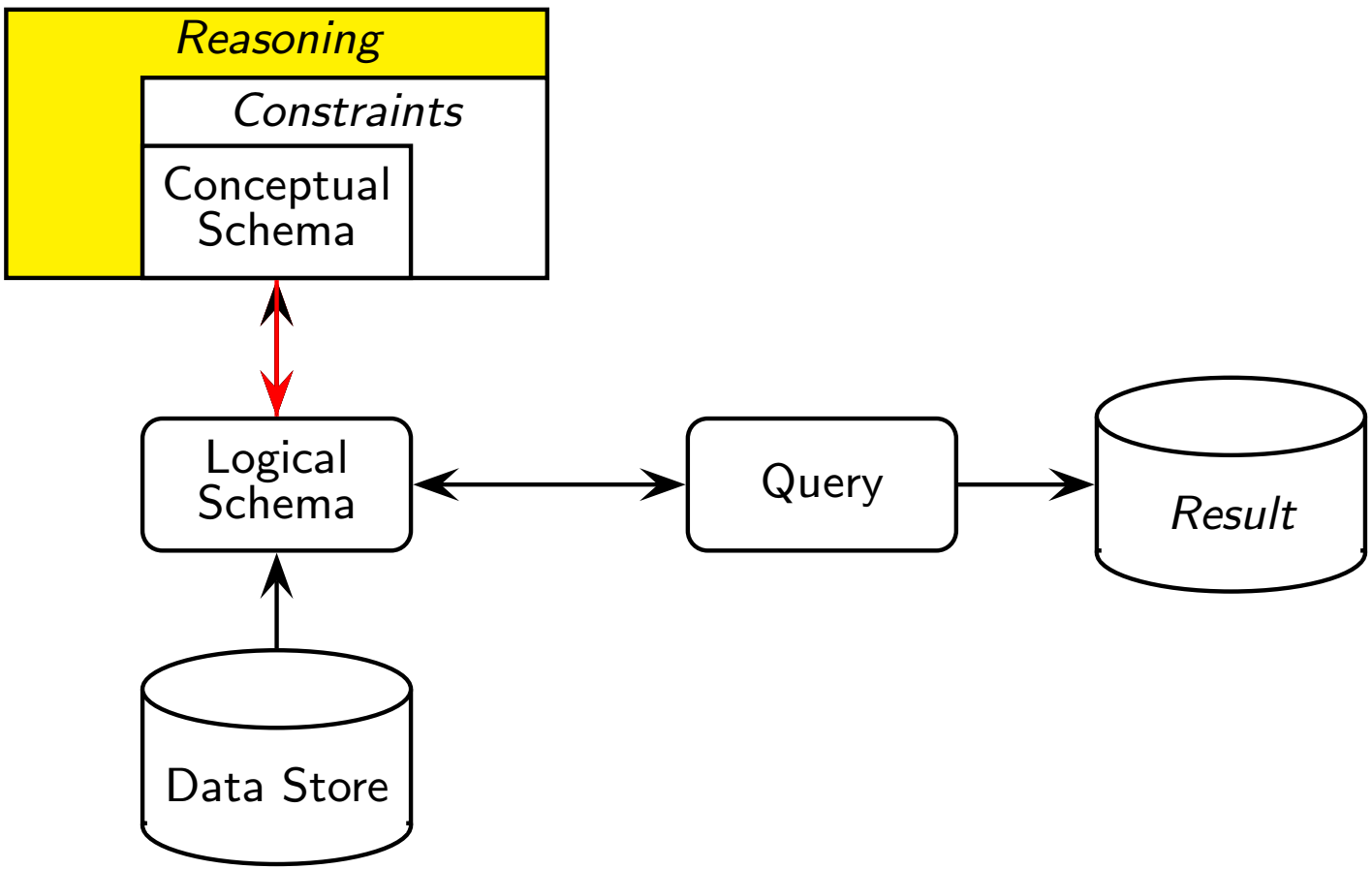
The role of a Conceptual Schema in Databases



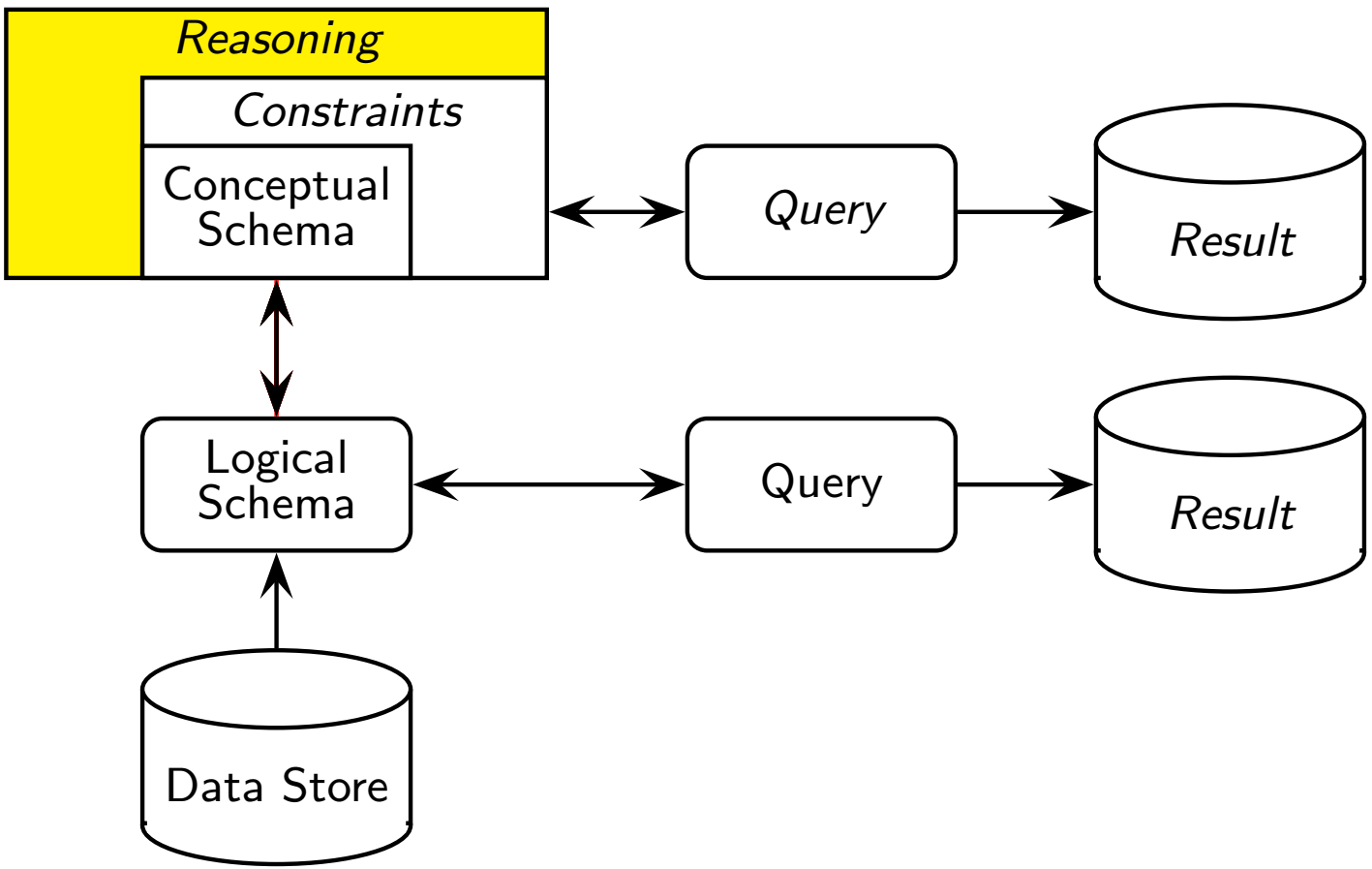
The role of a Conceptual Schema in Databases



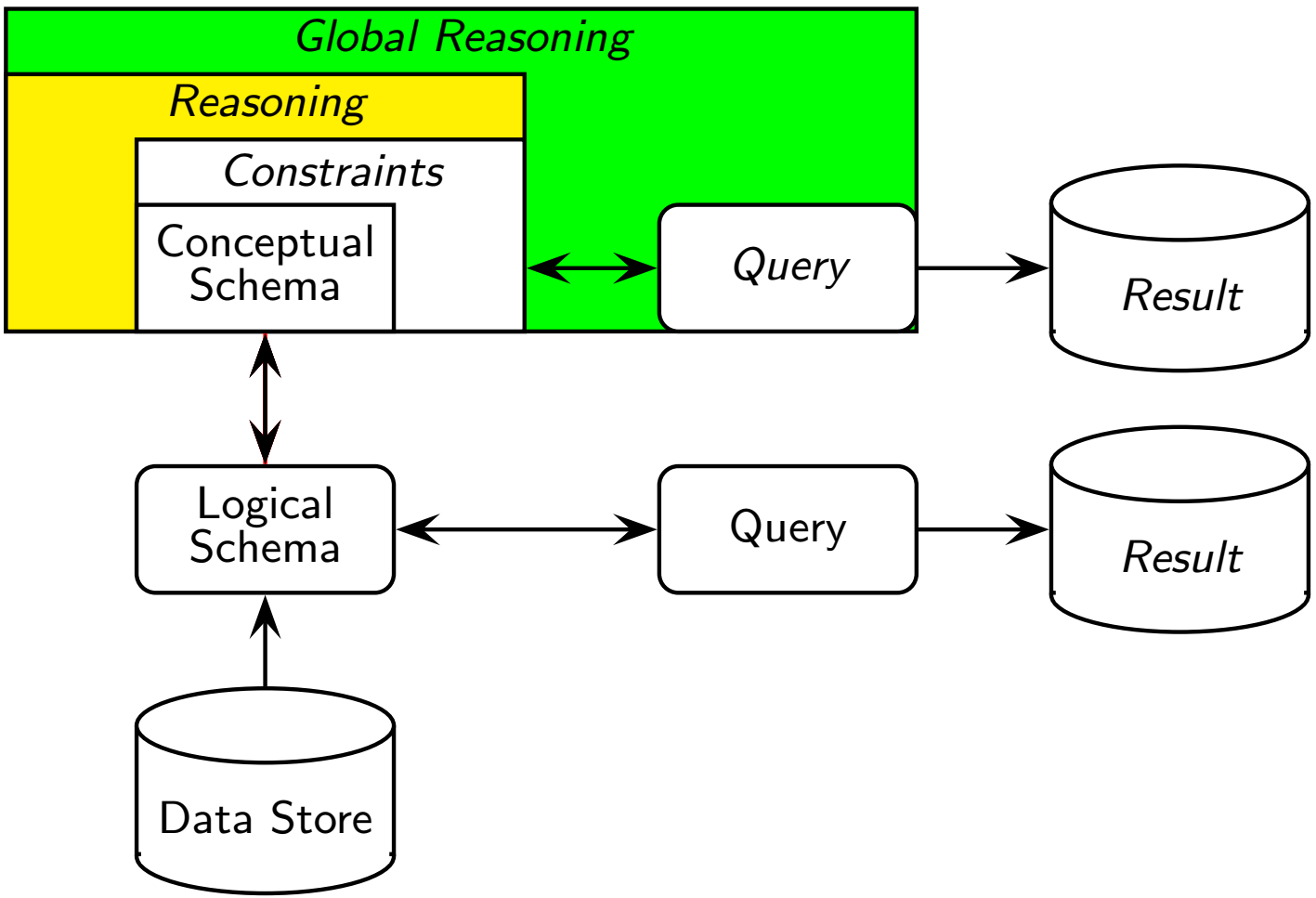
The role of a Conceptual Schema in Databases



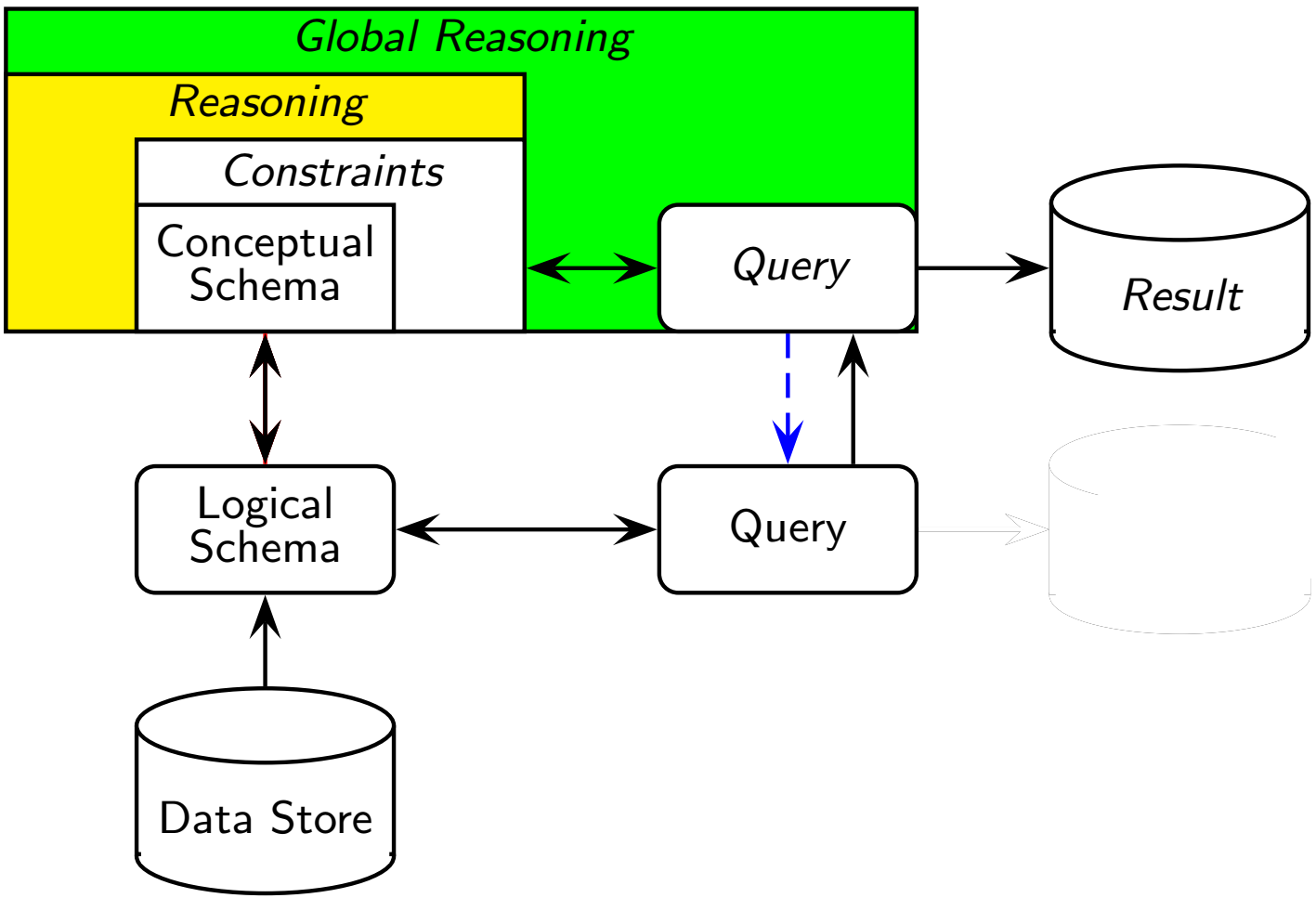
The role of a Conceptual Schema in Databases



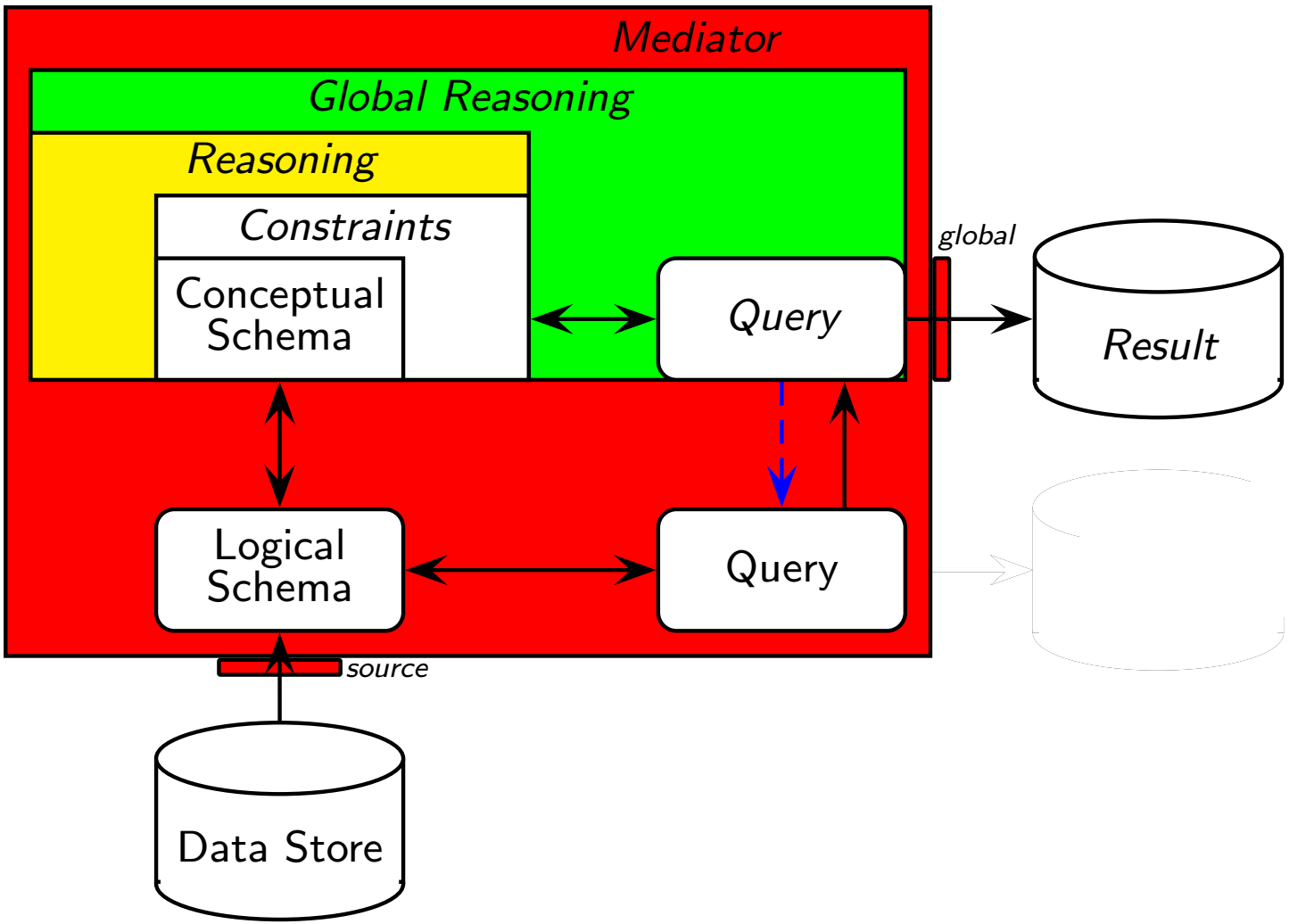
The role of a Conceptual Schema in Databases



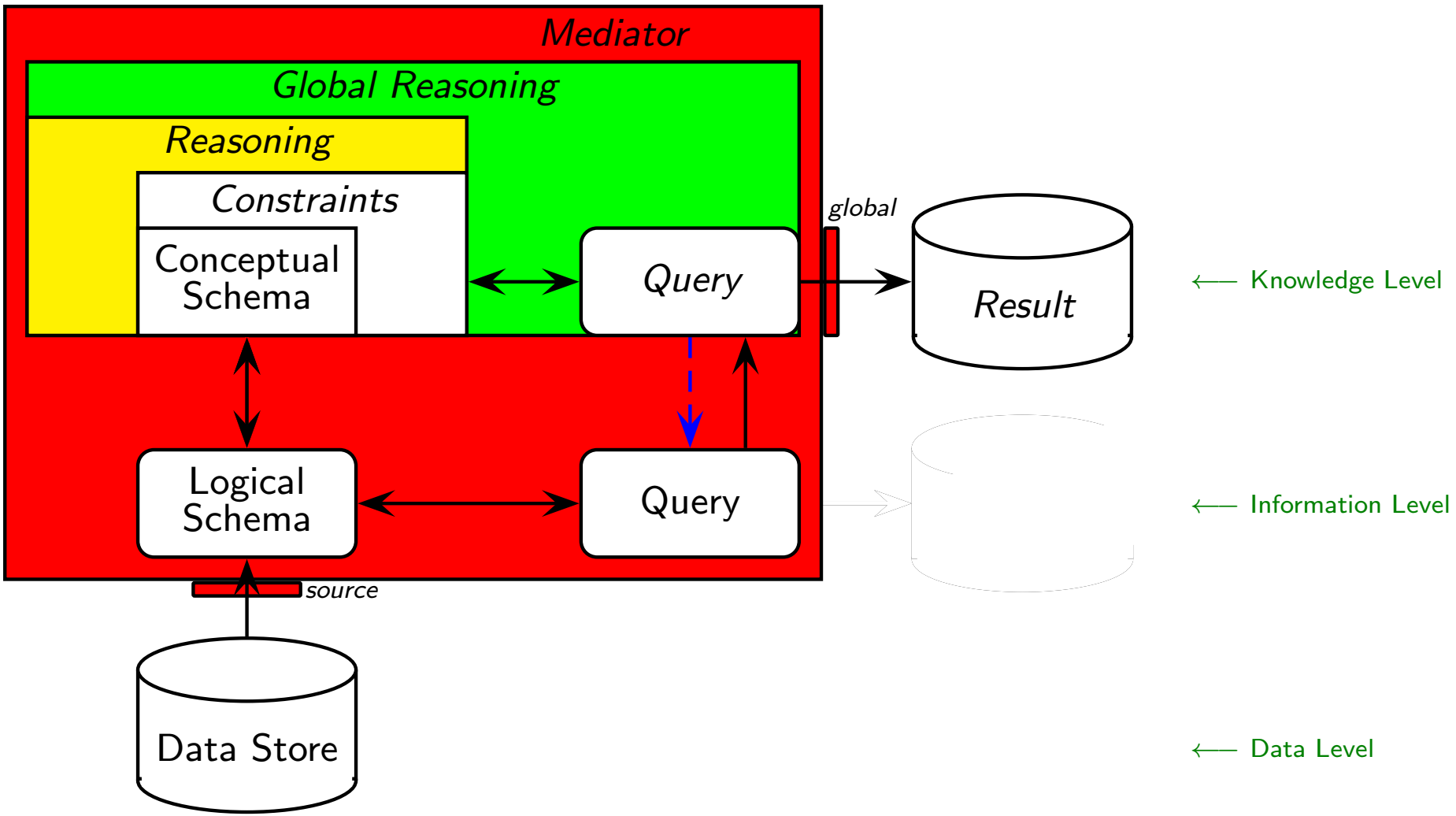
The role of a Conceptual Schema in Databases



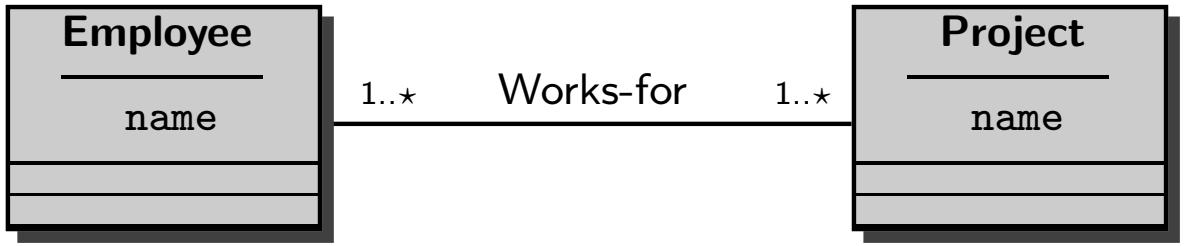
The role of a Conceptual Schema in Databases



The role of a Conceptual Schema in Databases



Conceptual Schemas with different vocabularies



Employee = { John, Mary}
EmployeeName = { ⟨John,"John"⟩, ⟨Mary,"Mary"⟩ }
Project = { Prj-A, Prj-B }
ProjectName = { ⟨Prj-A,"Prj-A"⟩, ⟨Prj-B,"Prj-B"⟩ }
Works-for = { ⟨John,Prj-A⟩, ⟨John,Prj-B⟩, ⟨Mary,Prj-A⟩ }

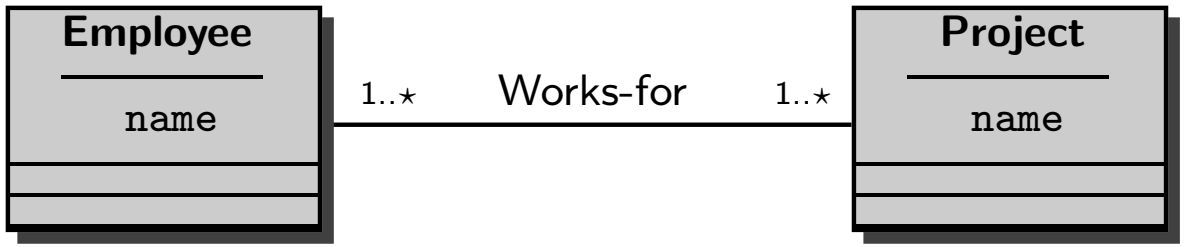
Employee-table:(EmpOID, Ename, ProjOID, Pname)
FD: ProjOID → Pname

Employee-table = {⟨John,"John", Prj-A,"Prj-A"⟩,
⟨John,"John", Prj-B,"Prj-B"⟩,
⟨mary,"Mary", Prj-A,"Prj-A"⟩}

Lossless Transformations (aka Equivalence)

- Two (conceptual) schemas with different vocabularies are lossless transformations of one another (i.e., they are equivalent) if there exist two total injective mappings from legal database instances in one schema to legal database instances in the other schema such that their composition is the identity.
- Query answering across equivalent schemas involves expanding those mappings as views.
- This notion is the formal foundation of Database Design, of Database Normalisation, and of Database Reverse Engineering.

Lossless Transformations



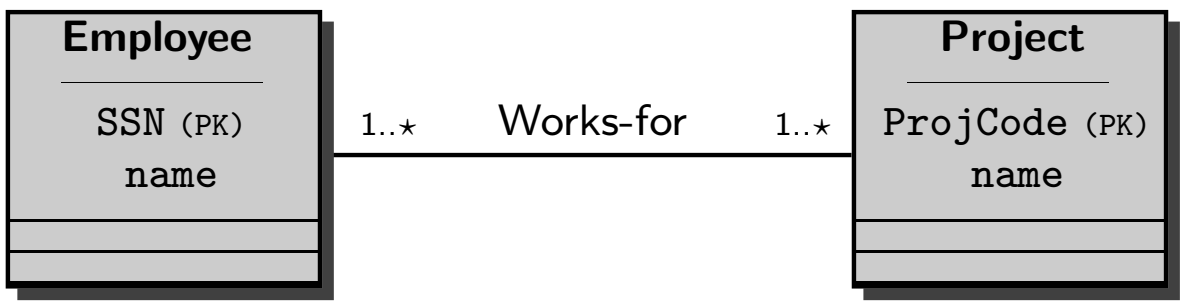
$\texttt{Employee} = \pi_{\texttt{EmpOID}}\ \texttt{Employee-table}$
$\texttt{EmployeeName} = \pi_{\texttt{EmpOID,Ename}}\ \texttt{Employee-table}$
$\texttt{Project} = \pi_{\texttt{ProjOID}}\ \texttt{Employee-table}$
$\texttt{ProjectName} = \pi_{\texttt{ProjOID,Pname}}\ \texttt{Employee-table}$
$\texttt{Works-for} = \pi_{\texttt{EmpOID,ProjOID}}\ \texttt{Employee-table}$

$\texttt{Employee-table} =$
$\quad \bowtie\ (\texttt{EmployeeName}, \texttt{ProjectName}, \texttt{Works-for})$

Employee-table:(<u>EmpOID</u>, Ename, <u>ProjOID</u>, Pname)
FD: ProjID $\rightarrow$ Pname

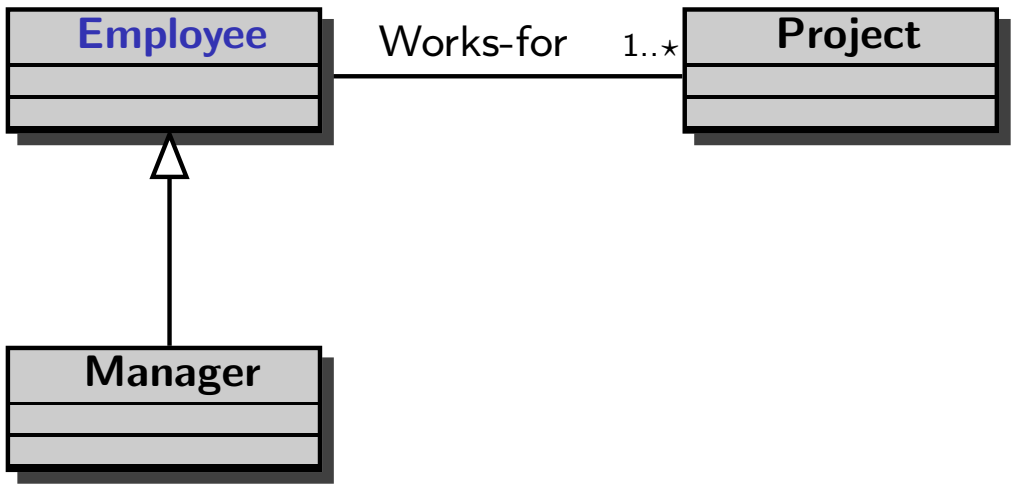
Object Identifiers

- A special lossless transformation: elimination of the Object Identifiers using Primary Keys or (better) Reference Scheme Modelling.



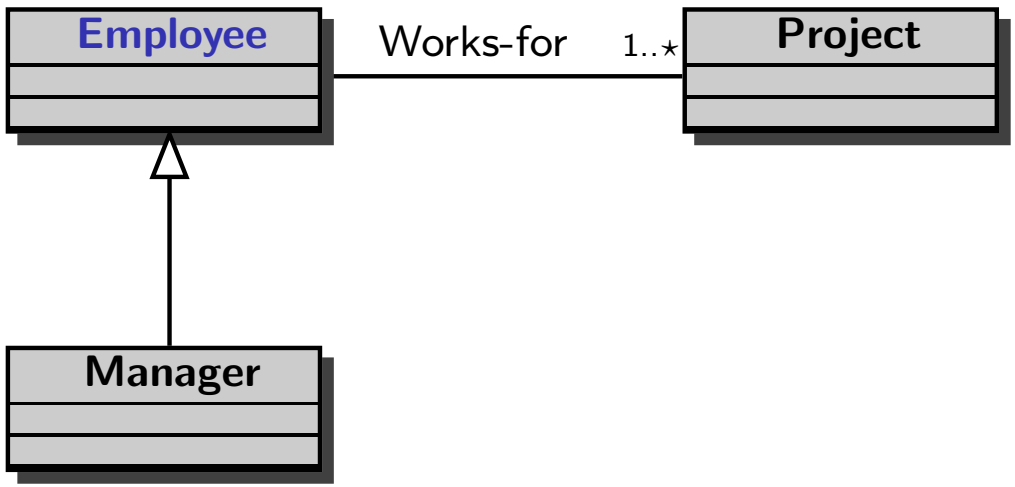
Employee-table:(SSN, Ename, ProjCode, Pname)
FD: ProjCode $\rightarrow$ Pname

Queries via Conceptual Schemas: the DBox case



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

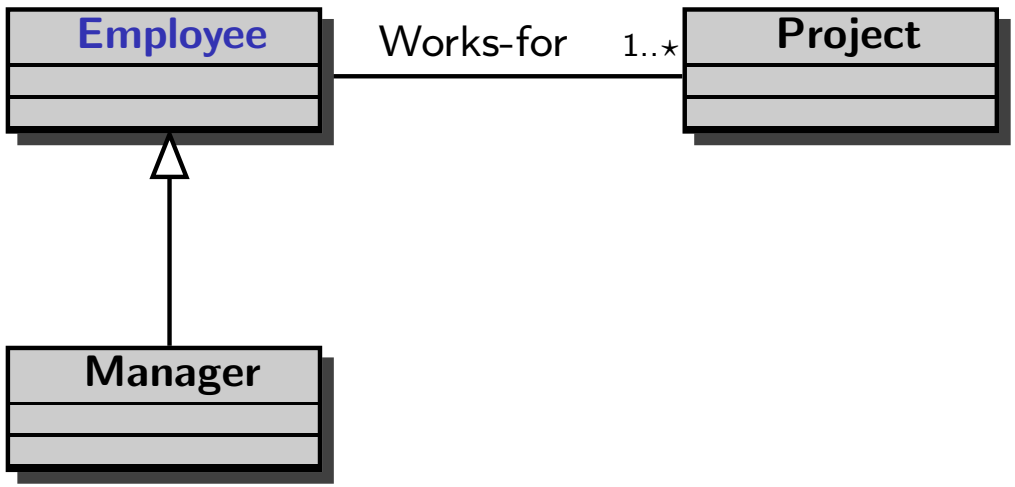
Queries via Conceptual Schemas: the DBox case



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

Q(x) :- Employee(x)

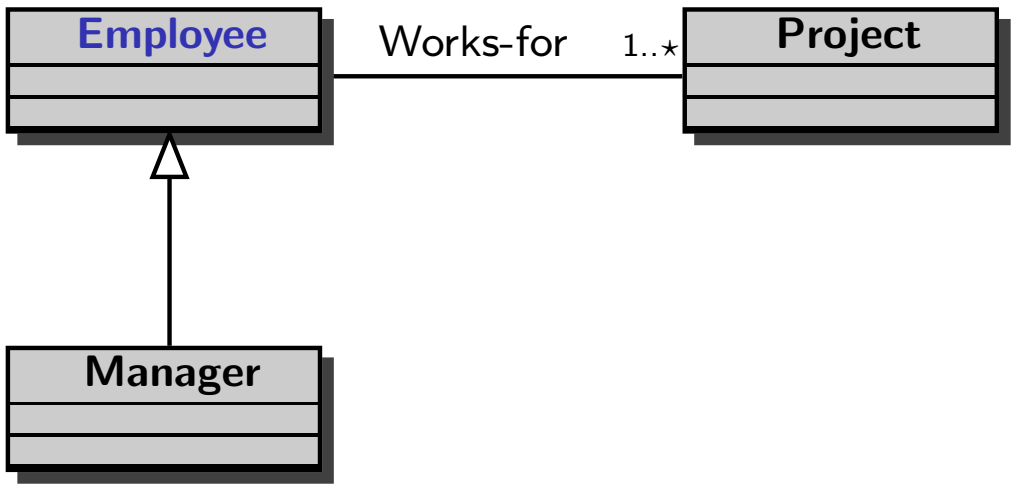
Queries via Conceptual Schemas: the DBox case



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

Q(x) :- Employee(x)
$\Longrightarrow$ { John, Paul, Mary } *certain answer*

Queries via Conceptual Schemas: the DBox case

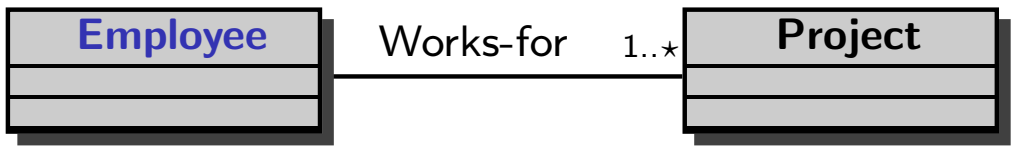


Manager = { John, Paul }
Works-for = { ⟨John,Prj-A⟩, ⟨Mary,Prj-B⟩ }
Project = { Prj-A, Prj-B }

Q(x) :- Employee(x)
$\Longrightarrow$ { John, Paul, Mary } *certain answer*

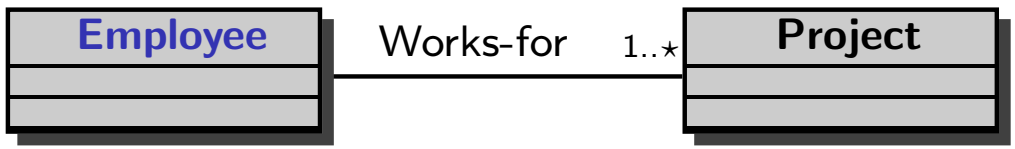
$\Longrightarrow$ Q'(x) :- Manager(x) $\vee$ $\exists$y. Works-for(x,y)

Queries via Conceptual Schemas: the ABox case



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

Queries via Conceptual Schemas: the ABox case

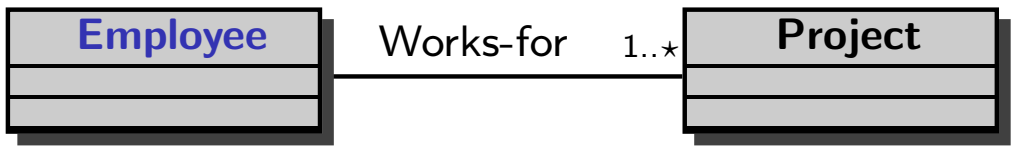


$\texttt{Works-for} \supseteq \{ \langle \texttt{John}, \texttt{Prj-A} \rangle, \langle \texttt{Mary}, \texttt{Prj-A} \rangle \}$

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

$\texttt{Q(y)} \texttt{ :- } \exists \texttt{x. Works-for(x,y)}$ *certain answer*

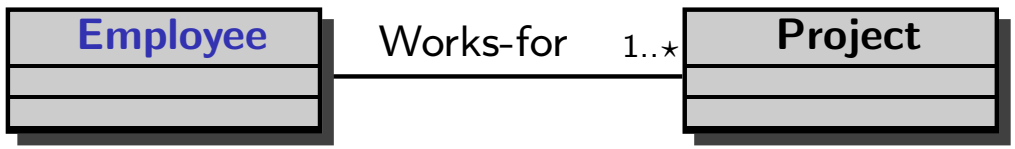
Queries via Conceptual Schemas: the ABox case



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

$\texttt{Q(y)} \texttt{ :- } \exists \texttt{x. Works-for(x,y)}$ *certain answer*
$\Longrightarrow \{ \texttt{Prj-A}, \texttt{Prj-B} \}$

Queries via Conceptual Schemas: the ABox case

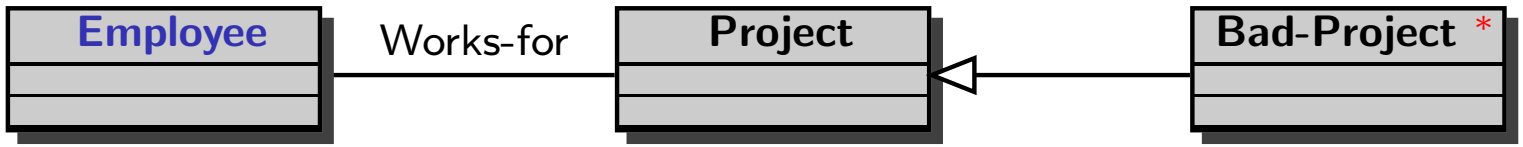


Works-for $\supseteq$ { ⟨John,Prj-A⟩, ⟨Mary,Prj-A⟩ }
Project $\supseteq$ { Prj-A, Prj-B }

Q(y) :- ∃x. Works-for(x,y) *certain answer*
$\Longrightarrow$ { Prj-A, Prj-B }

$\Longrightarrow$ Q'(y) :- Project(y) ∨ ∃x. Works-for(x,y)

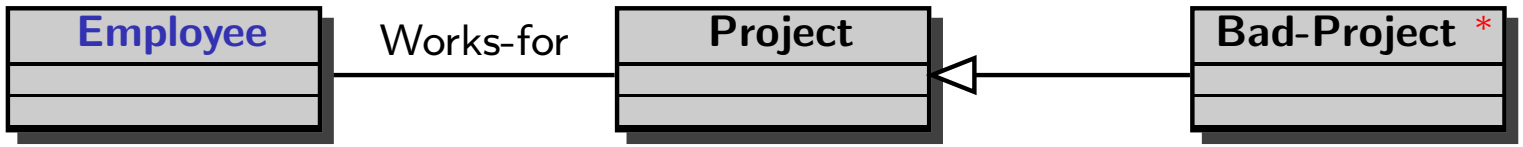
An ABox changes the DB semantics



`Bad-Project` defined with a rule/view:

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

An ABox changes the DB semantics



Bad-Project defined with a rule/view:

* Bad-Project(x) $\leftrightarrow$ Project(x)$\wedge$ $\neg\exists$y. Works-for(y,x)

- DBox:

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

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

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

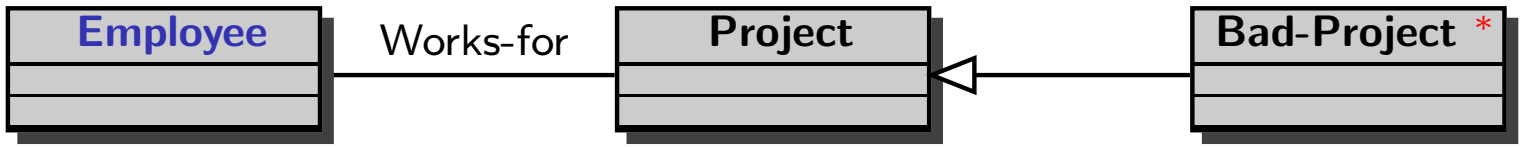
- ABox:

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

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

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

An ABox changes the DB semantics



Bad-Project defined with a rule/view:

* Bad-Project(x) $\leftrightarrow$ Project(x)$\wedge$ $\neg\exists$y. Works-for(y,x)

- DBox:

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

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

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

  $\Longrightarrow$ { Prj-B }

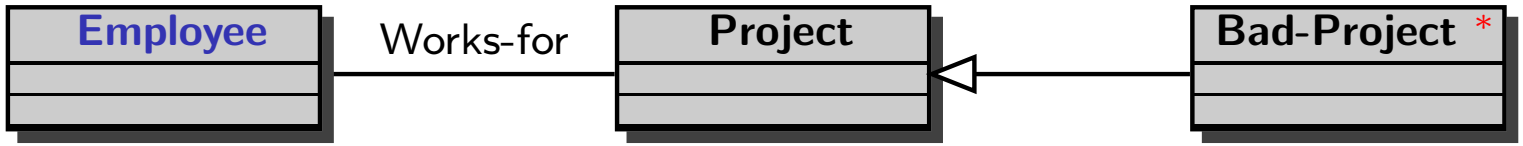
- ABox:

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

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

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

An ABox changes the DB semantics



Bad-Project defined with a rule/view:

* Bad-Project(x) $\leftrightarrow$ Project(x)$\wedge$ $\neg\exists$y. Works-for(y,x)

- DBox:

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

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

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

  $\Longrightarrow$ { Prj-B }

- ABox:

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

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

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

  $\Longrightarrow$ { }

Conclusions

Conclusions

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

Conclusions

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

Pay attention!