Knowledge Representation and Databases

Enrico Franconi

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

http://krdb.eu/franconi

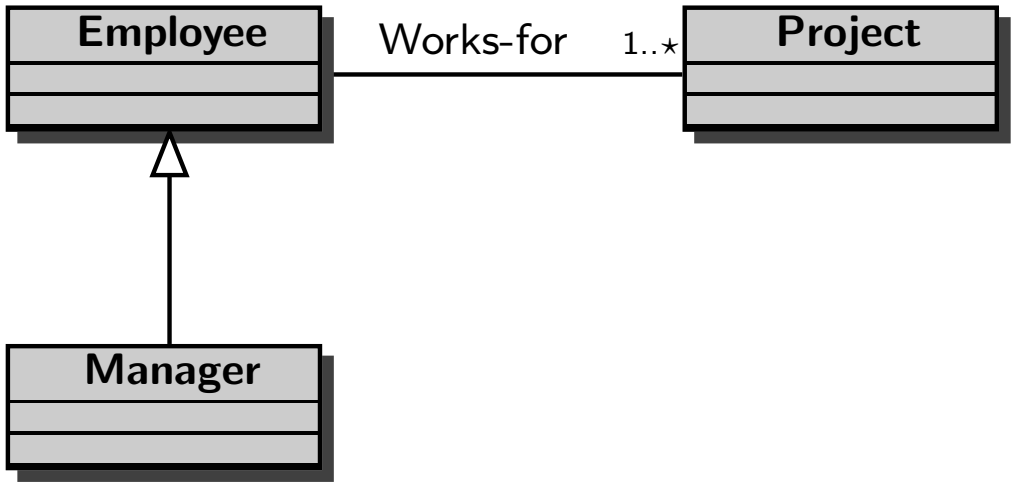
Cape-KR 2024

Summary of this talk

- Databases and Logic
- Conceptual Schemas
- Querying databases via the conceptual schema
- DBoxes and ABoxes
- Conceptual Schemas with different vocabularies
- Lossless schema transformations
- Query rewriting
- Null values in SQL

Databases and Logic

- A database is a finite relational structure (a first-order interpretation)
- Database constraints are first-order logic formulas; a database should be a model of these constraints
- $\rightarrow$ satisfaction
- $\rightarrow$ querying



$$\texttt{Employee} = \{\ \texttt{John}, \texttt{Mary}, \texttt{Paul}\ \}$$
$$\texttt{Manager} = \{\ \texttt{John}, \texttt{Paul}\ \}$$
$$\texttt{Works-for} = \{\ \langle \texttt{John}, \texttt{Prj-A} \rangle, \langle \texttt{Mary}, \texttt{Prj-B} \rangle\ \}$$
$$\texttt{Project} = \{\ \texttt{Prj-A}, \texttt{Prj-B}\ \}$$

Databases and Logic

- A database is a finite relational structure (a first-order interpretation)
- Database constraints are first-order logic formulas; a database should be a model of these constraints
- $\rightarrow$ satisfaction
- $\rightarrow$ querying

- An incomplete database is a set of databases
- An incomplete database can be expressed by the models of a logic formula
- $\rightarrow$ entailment
- $\rightarrow$ querying

Databases and Logic

- A database is a finite relational structure (a first-order interpretation)
- Database constraints are first-order logic formulas; a database should be a model of these constraints
- $\rightarrow$ satisfaction
- $\rightarrow$ querying

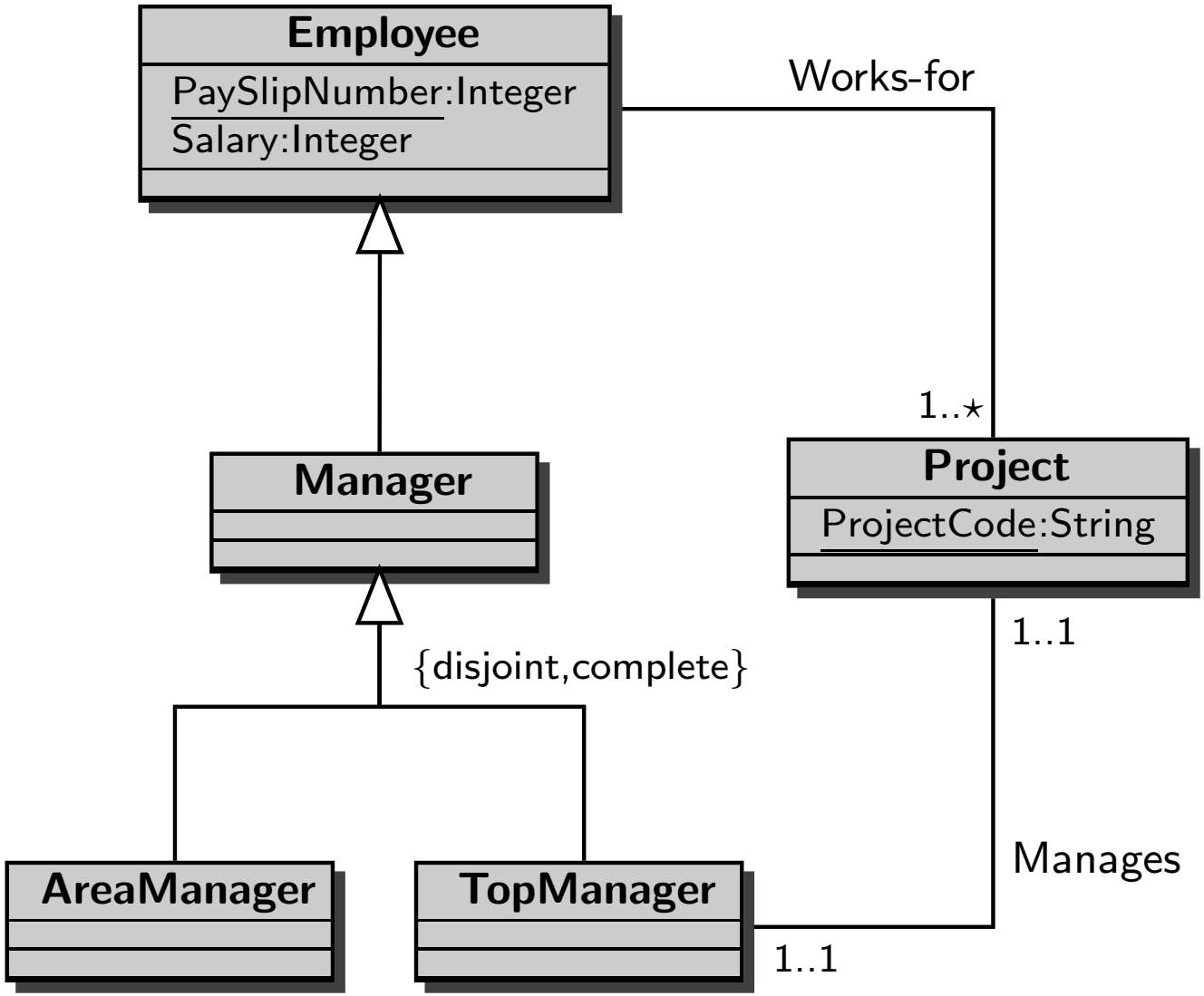
- An incomplete database is a set of databases
- An incomplete database can be expressed by the models of a logic formula
- $\rightarrow$ entailment
- $\rightarrow$ querying

- (By the way, also a single database can be expressed by means of logic formulas $\rightarrow$ Reiter)

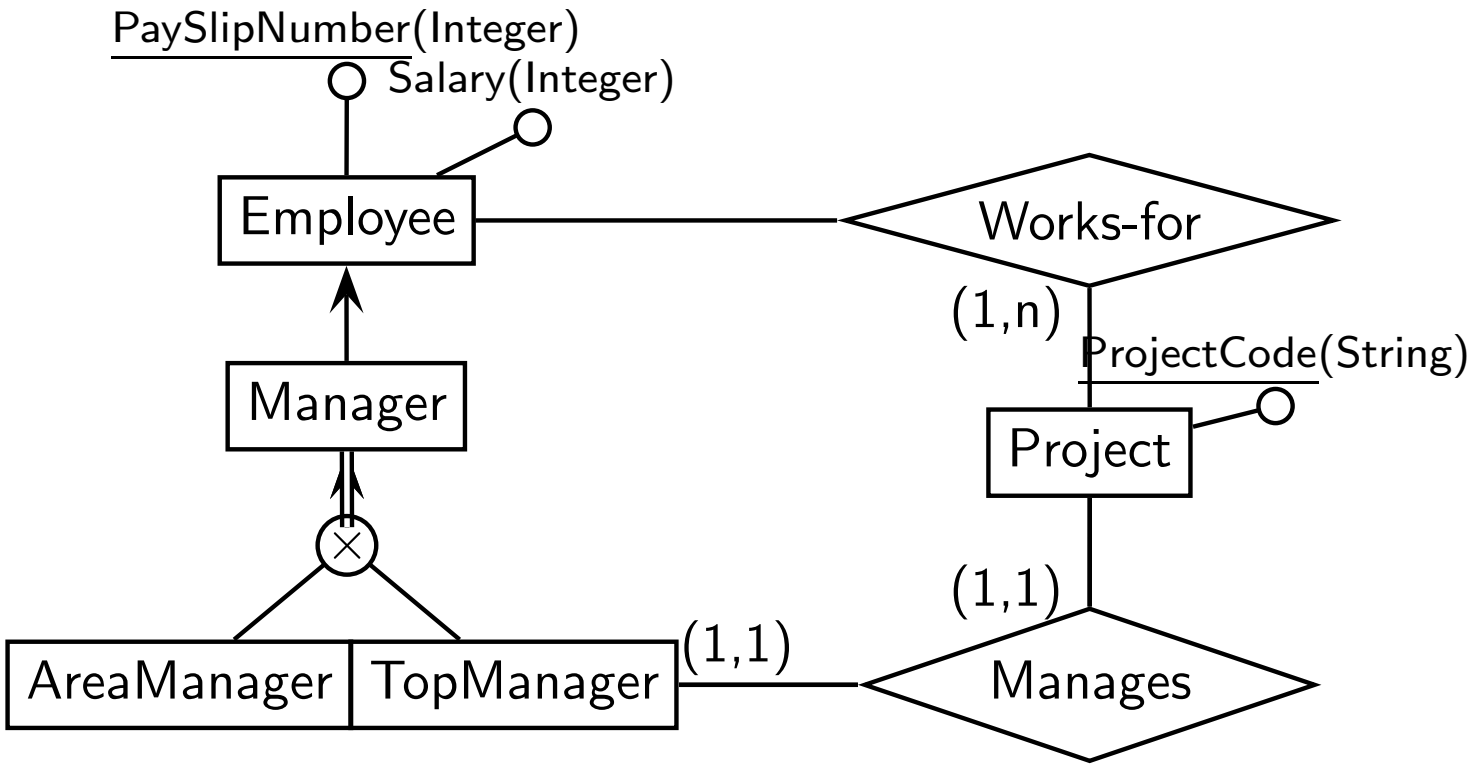
Ontologies (Conceptual Schemas)

- 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.
- An ontology is a semantic layer including a set of first order constraints (the *semantics*) over the given vocabulary, which specifies what should hold in any possible configuration of the system.
- Given an ontology, a *legal database* is a (finite) model satisfying the constraints.

UML Class Diagram



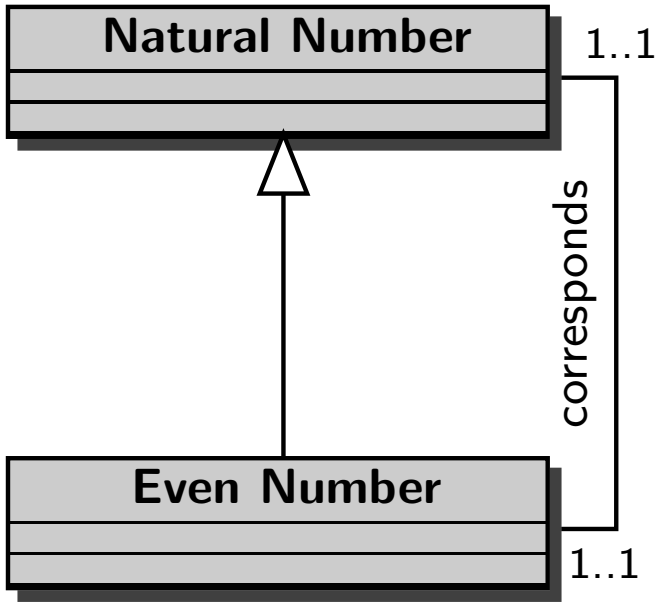
Entity-Relationship Schema



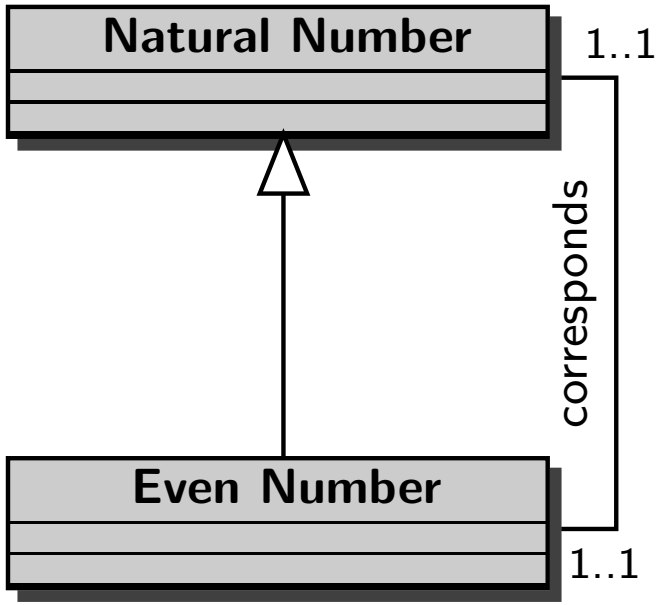
Constraints induced by the diagram

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

Bijection: how many numbers

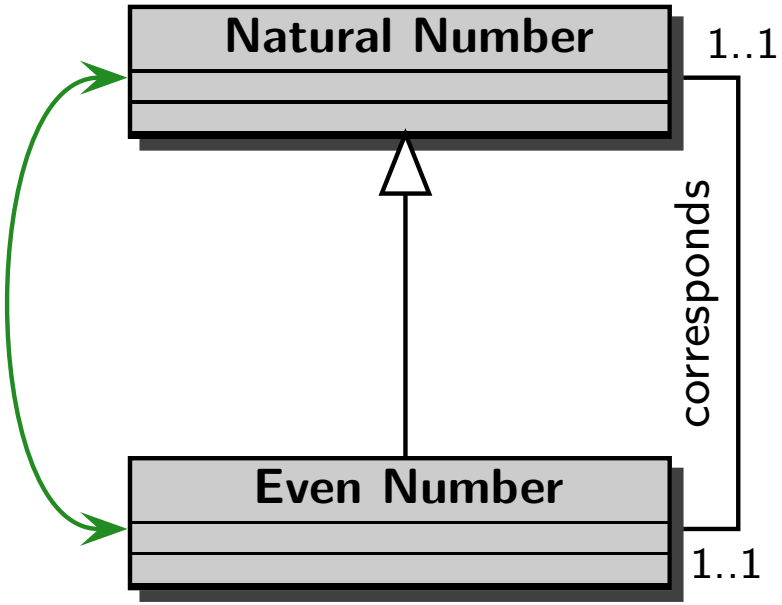


Bijection: how many numbers



the classes *'Natural Number'* and *'Even Number'* contain the same number of instances

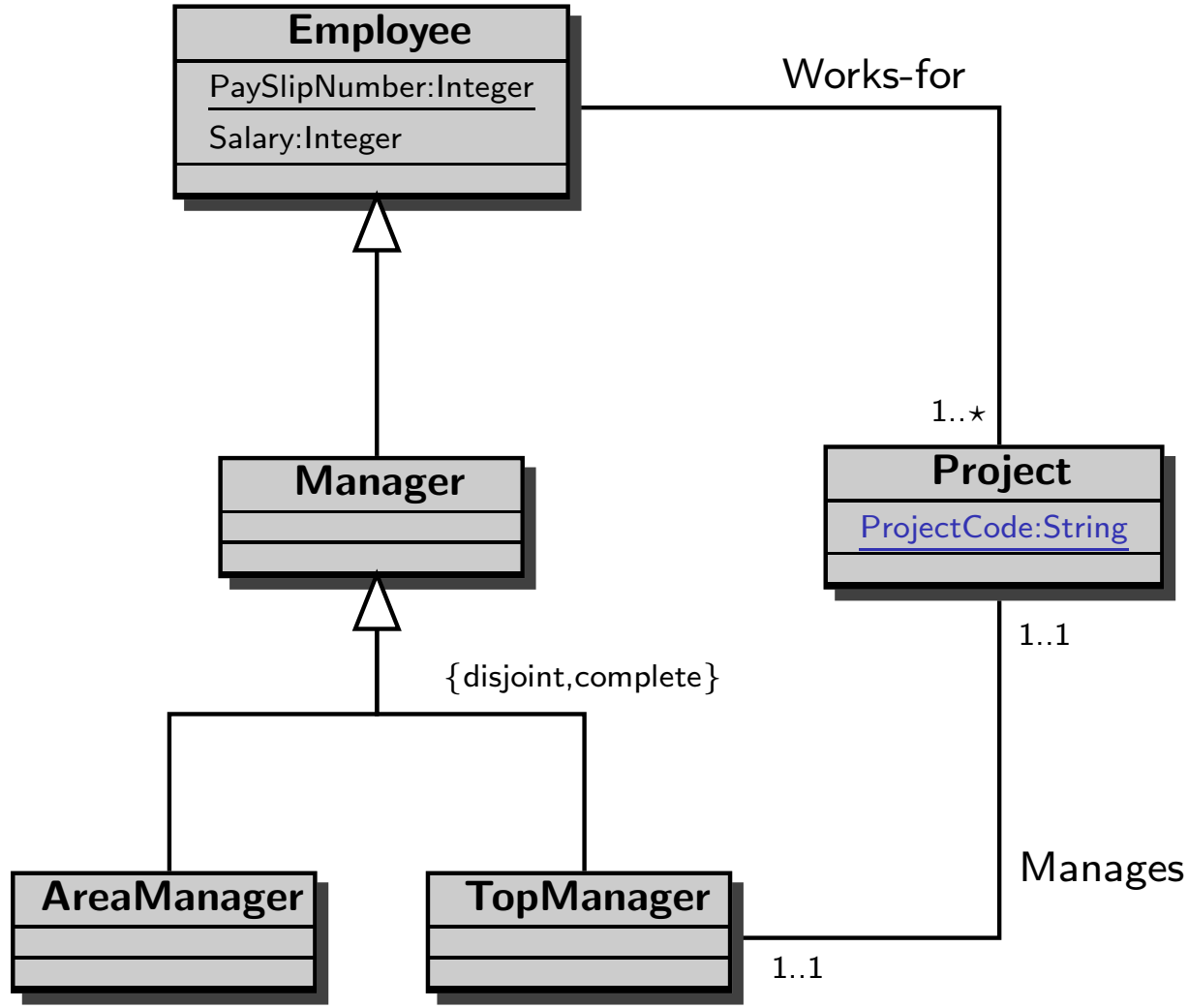
Bijection: how many numbers



the classes *'Natural Number'* and *'Even Number'*
contain the same number of instances

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

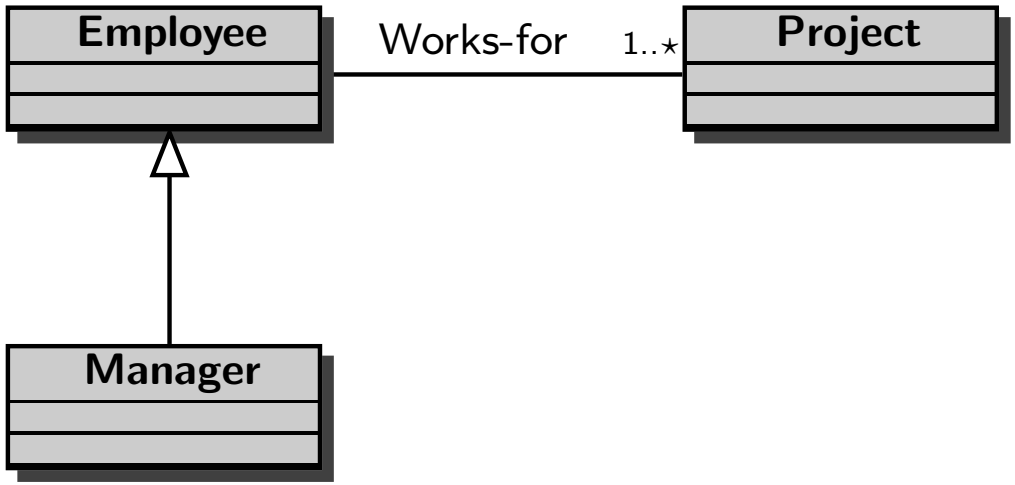
Key constraints



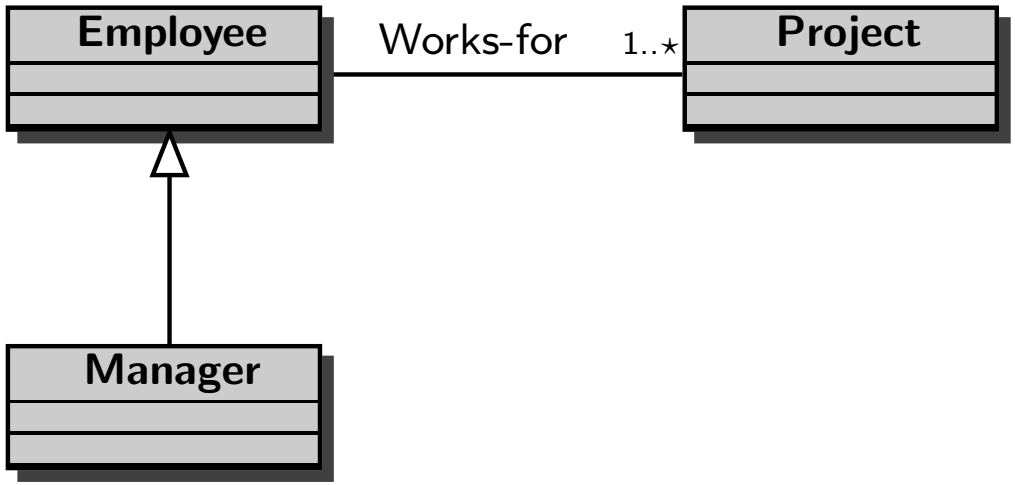
$\forall \mathtt{x}.\ \mathtt{Project}(\mathtt{x}) \rightarrow \exists^{=1}\mathtt{y}.\ \mathtt{ProjectCode}(\mathtt{x},\mathtt{y}) \wedge \mathtt{String}(\mathtt{y})$

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

Queries via Conceptual Schemas: the DB case

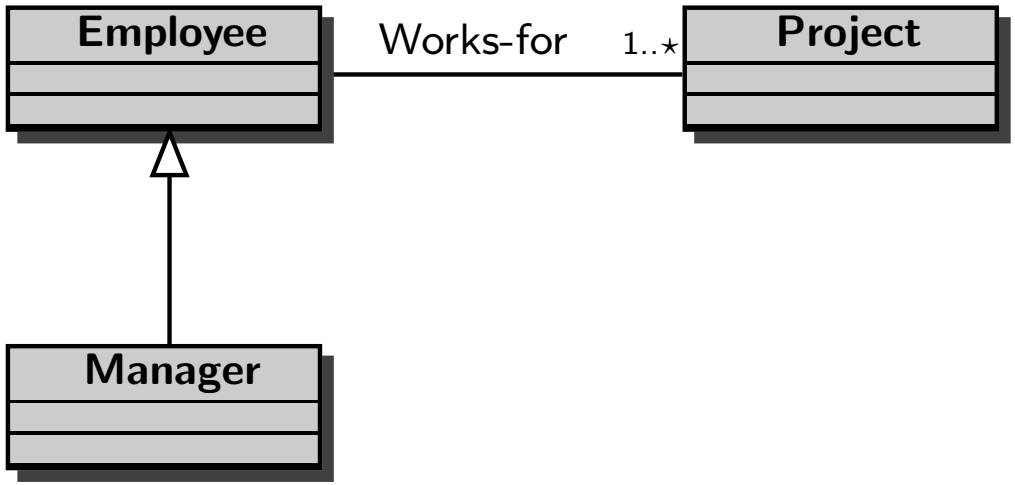


Queries via Conceptual Schemas: the DB case



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

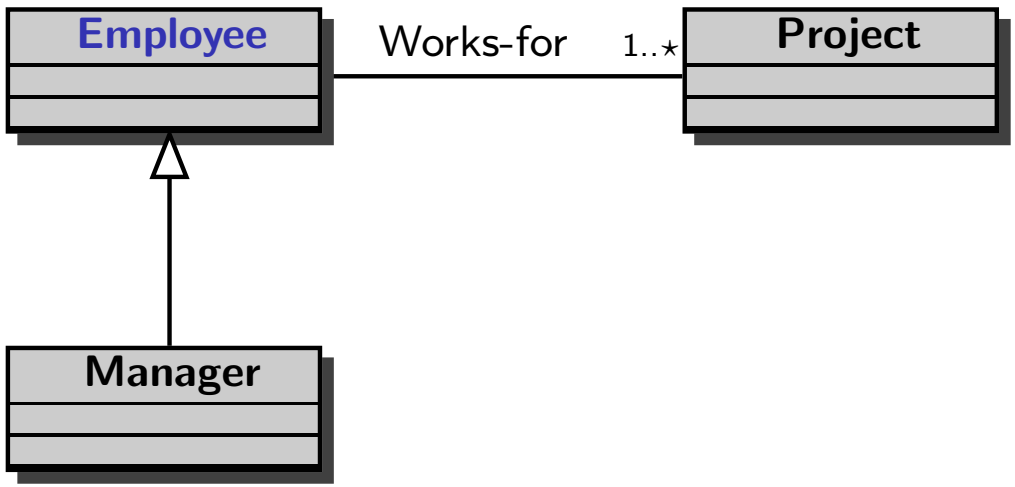
Queries via Conceptual Schemas: the DB case



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

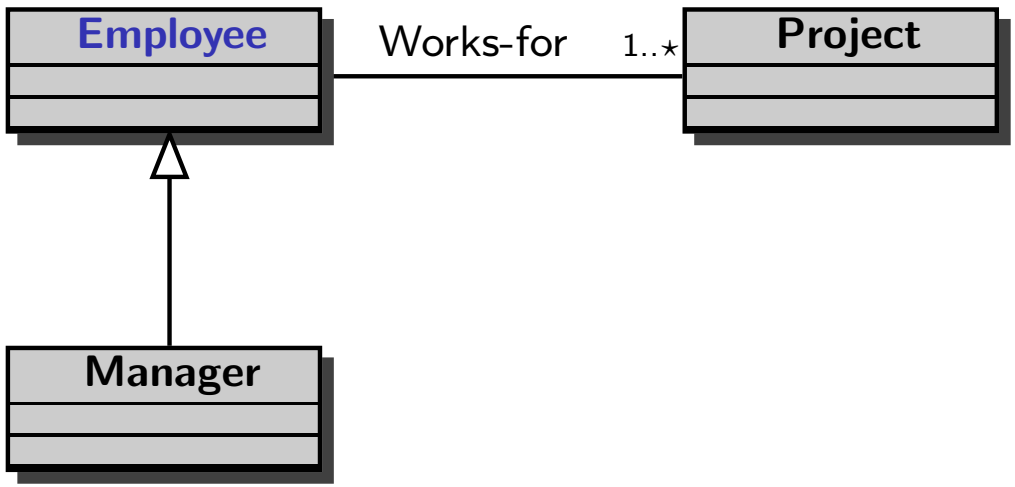
Q(x) :- ∃y. Manager(x) ∧ Works-for(x,y) ∧ Project(y)
⟹ { John }

Queries via Conceptual Schemas: the DBox case



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

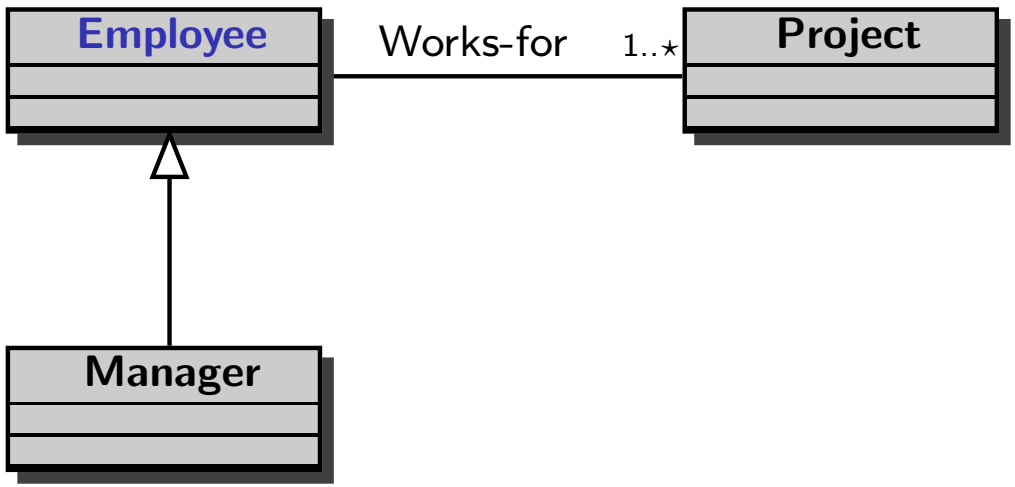
Queries via Conceptual Schemas: the DBox case



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

Q(x) :- Employee(x)

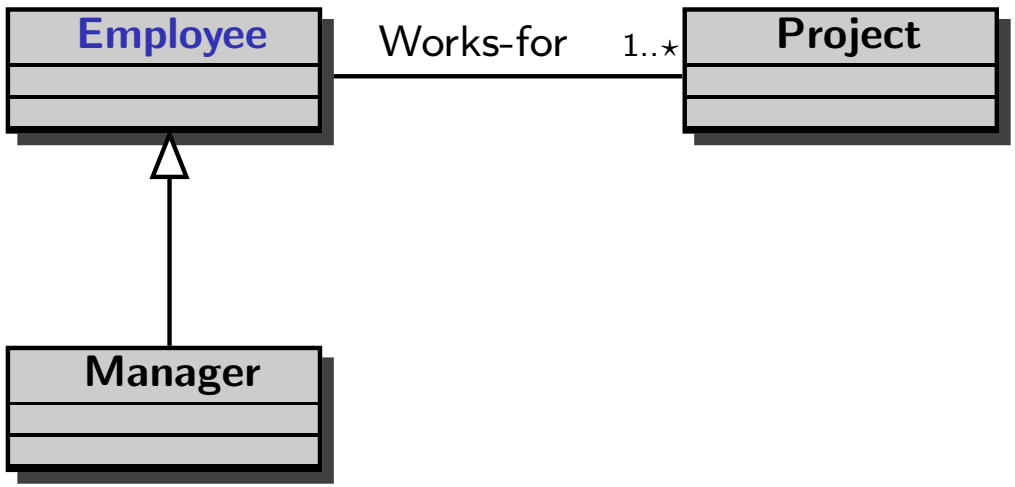
Queries via Conceptual Schemas: the DBox case



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* (entailment)

Queries via Conceptual Schemas: the DBox case

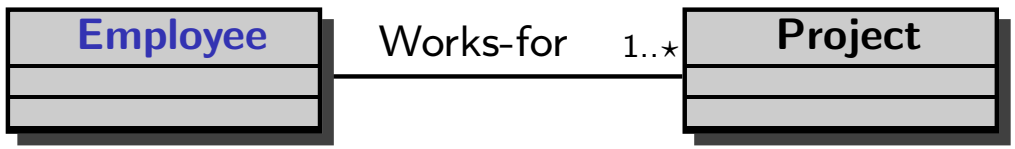


Manager = { John, Paul }
Works-for = { ⟨John,Prj-A⟩, ⟨Mary,Prj-B⟩ }
Project = { Prj-A, Prj-B }

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

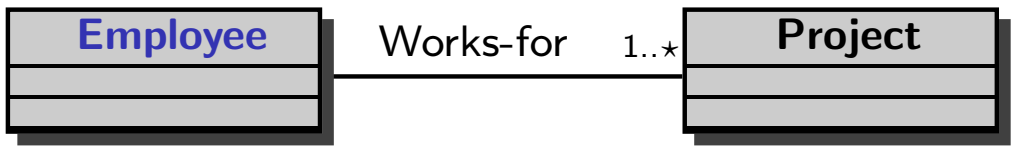
$\Longrightarrow$ Q'(x) :- Manager(x) $\vee$ $\exists$y. Works-for(x,y)

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

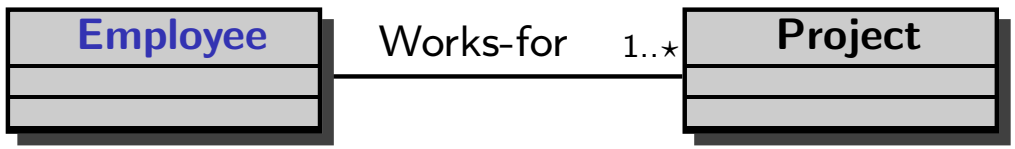


$\texttt{Works-for} \supseteq \{ \langle \texttt{John}, \texttt{Prj-A} \rangle, \langle \texttt{Mary}, \texttt{Prj-A} \rangle \}$

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

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

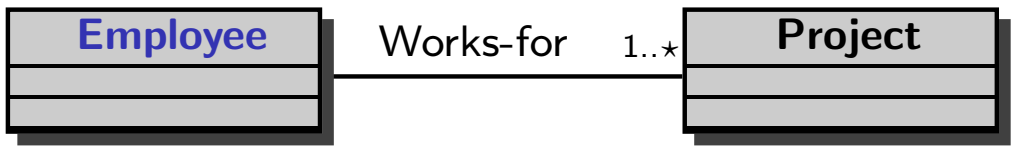
Queries via Conceptual Schemas: the ABox case



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

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

Queries via Conceptual Schemas: the ABox case



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

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

$\Longrightarrow \quad \texttt{Q'(y)} \texttt{ :- Project(y)} \vee \exists \texttt{x. Works-for(x,y)}$

Bad news

Query answering with certain answer semantics
with DBoxes or ABoxes
with expressive conceptual modelling languages
is coNP-hard in data complexity.

Bad news

Query answering with certain answer semantics
with DBoxes or ABoxes
with expressive conceptual modelling languages
is coNP-hard in data complexity.

Fixes:

1. reduce expressivity of conceptual modelling languages and queries (OBDA approach);
2. use exact answer semantics allowing only determined queries (reduction to lossless transformations).

Bad news

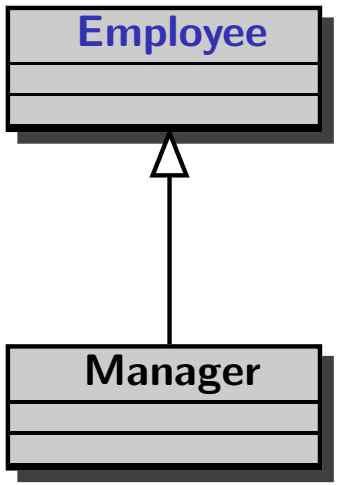
Query answering with certain answer semantics
with DBoxes or ABoxes
with expressive conceptual modelling languages
is coNP-hard in data complexity.

Fixes:

1. reduce expressivity of conceptual modelling languages and queries (OBDA approach);
2. use exact answer semantics allowing only determined queries (reduction to lossless transformations).

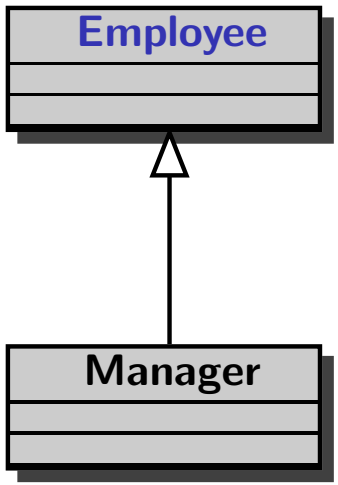
More bad news follow.

Queries with certain answer semantics



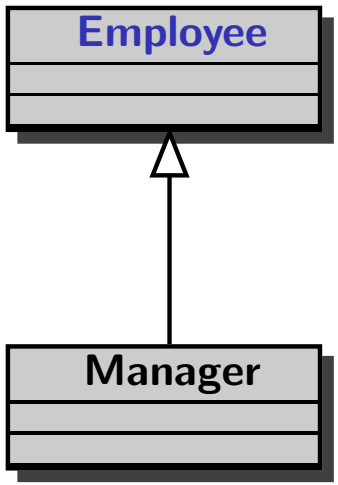
Manager = { John, Paul }

Queries with certain answer semantics




```
Manager = { John, Paul }

Q(x) :- Employee(x)
```

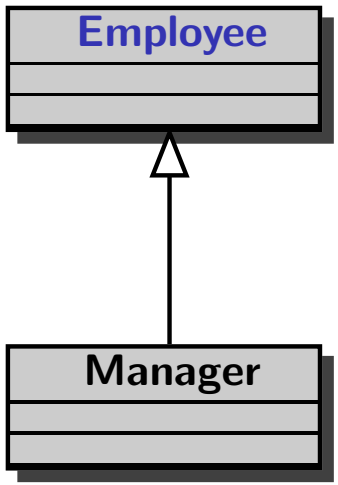
Queries with certain answer semantics



`Manager = { John, Paul }`

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

Queries with certain answer semantics

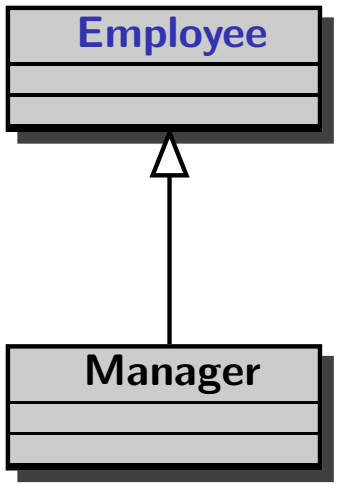


`Manager` = { `John`, `Paul` }

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

So: are `Managers` and `Employees` the same?

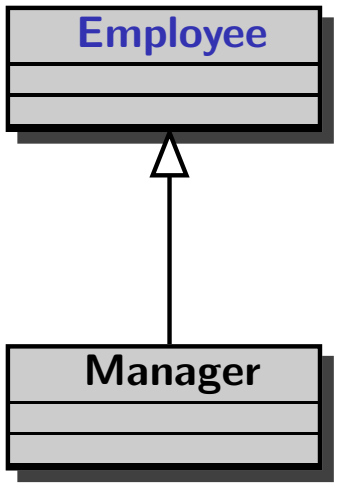
Queries with certain answer semantics



Manager = { John, Paul }

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

Queries with certain answer semantics



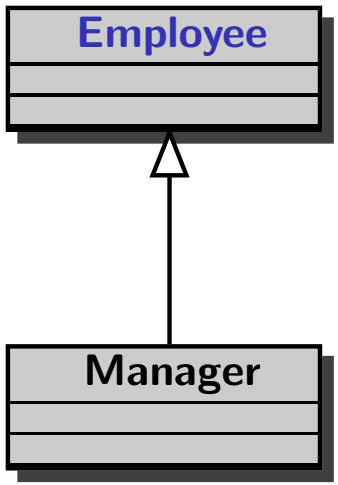
Manager = { John, Paul }

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

Q^1 :- Manager(George)

Q^2 :- Employee(George)

Queries with certain answer semantics



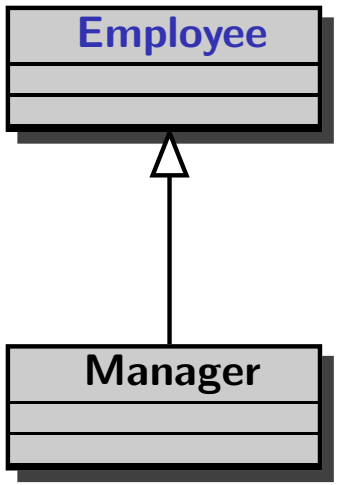
Manager = { John, Paul }

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

Q^1 :- Manager(George) $\Longrightarrow$ FALSE

Q^2 :- Employee(George)

Queries with certain answer semantics



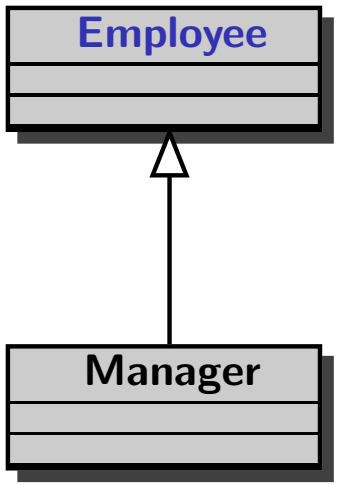
Manager = { John, Paul }

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

Q^1 :- Manager(George) $\Longrightarrow$ FALSE

Q^2 :- Employee(George) $\Longrightarrow$ DON'T KNOW

Queries with certain answer semantics



`Manager = { John, Paul }`

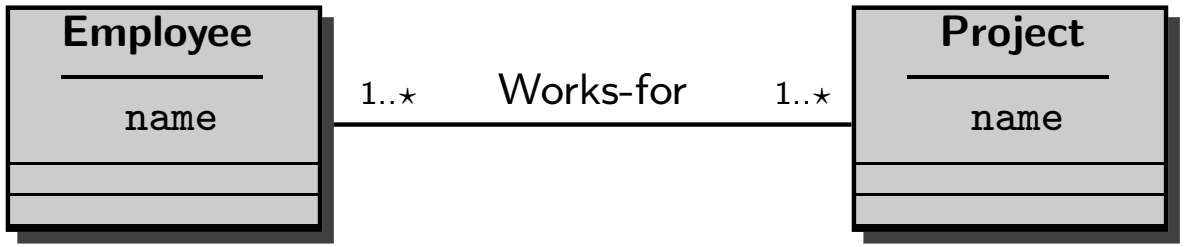
`Q(x) :- Employee(x)` $\Longrightarrow$ `{ John, Paul }` *certain answer*

Q^1 `:- Manager(George)` $\Longrightarrow$ FALSE

Q^2 `:- Employee(George)` $\Longrightarrow$ DON'T KNOW

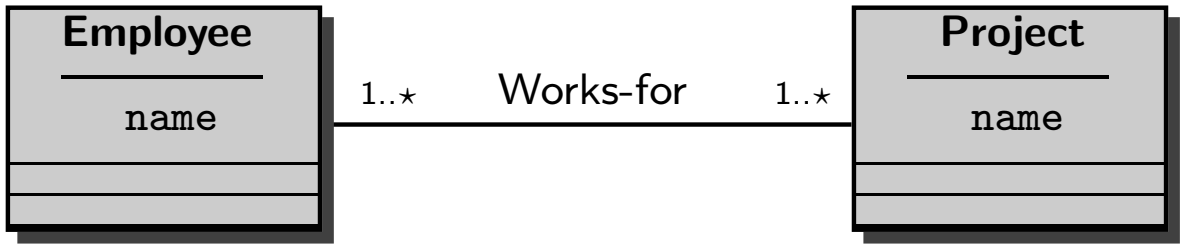
The result of the query can not be materialised (as a view)

The general case:
Conceptual Schemas with different vocabularies



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

The general case:
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⟩ }

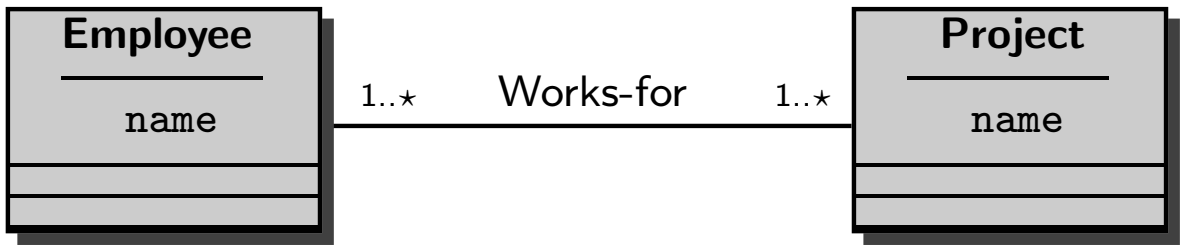
Employee-table:(EmpOID, Ename, ProjOID, Pname)
FD: EmpOID → Ename
FD: ProjOID → Pname

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 and updates across equivalent schemas involves expanding those mappings as views.

Lossless Transformations

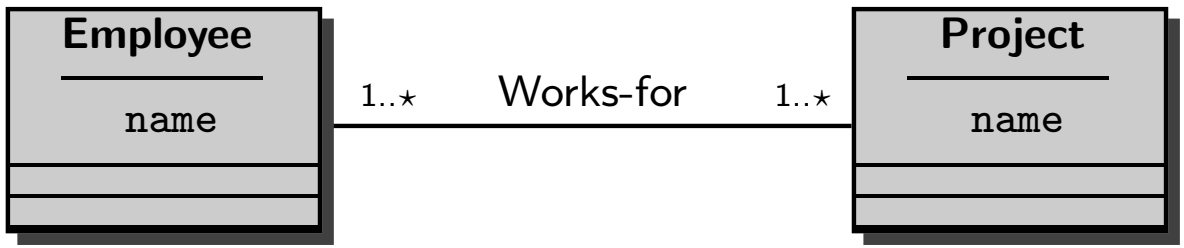


Employee-table:(EmpOID, Ename, ProjOID, Pname)

FD: EmpOID $\rightarrow$ EnameD

FD: ProjOID $\rightarrow$ Pname

Lossless Transformations



$\texttt{Employee} = \pi_{\texttt{EmpOID}}\ \texttt{Employee-table}$

$\texttt{EmployeeName} = \pi_{\texttt{EmpOID,Ename}}\ \texttt{Employee-table}$

$\texttt{Project} = \pi_{\texttt{ProjOID}}\ \texttt{Employee-table}$

$\texttt{ProjectName} = \pi_{\texttt{ProjOID,Pname}}\ \texttt{Employee-table}$

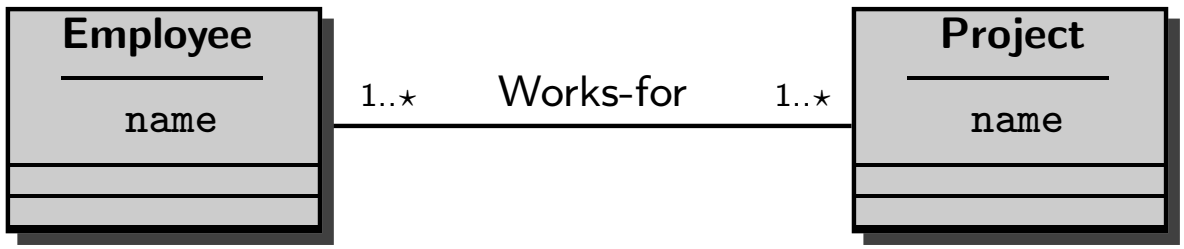
$\texttt{Works-for} = \pi_{\texttt{EmpOID,ProjOID}}\ \texttt{Employee-table}$

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

FD: EmpOID $\rightarrow$ EnameD

FD: ProjOID $\rightarrow$ Pname

Lossless Transformations



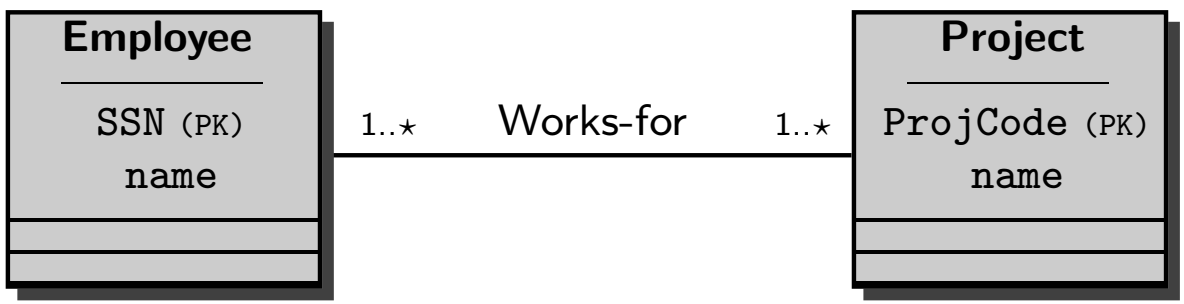
$\texttt{Employee} = \pi_{\texttt{EmpOID}}\ \texttt{Employee-table}$
$\texttt{EmployeeName} = \pi_{\texttt{EmpOID,Ename}}\ \texttt{Employee-table}$
$\texttt{Project} = \pi_{\texttt{ProjOID}}\ \texttt{Employee-table}$
$\texttt{ProjectName} = \pi_{\texttt{ProjOID,Pname}}\ \texttt{Employee-table}$
$\texttt{Works-for}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

$$\texttt{Employee-table} = \texttt{EmployeeName} \bowtie \texttt{ProjectName} \bowtie \texttt{Works-for}$$

Employee-table:(EmpOID, Ename, ProjOID, Pname)
FD: EmpOID $\rightarrow$ EnameD
FD: ProjOID $\rightarrow$ Pname

Object Identifiers

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



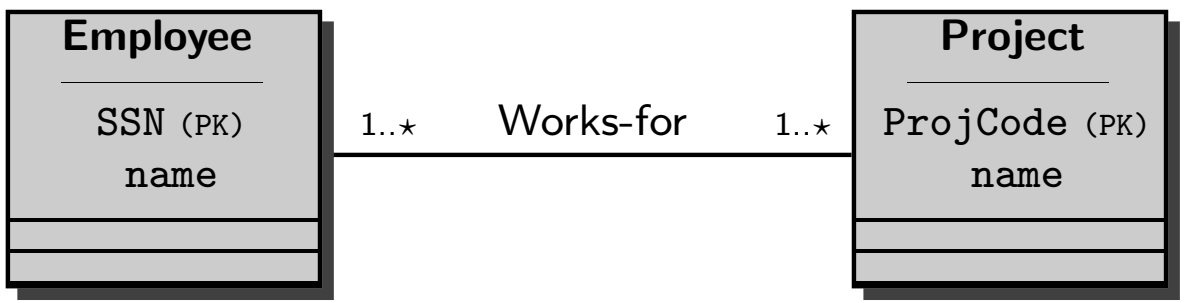
Employee-table:(SSN, Ename, ProjCode, Pname)
FD: EmpOID $\rightarrow$ EnameD
FD: ProjOID $\rightarrow$ Pname

The Canonical Abstract Relational Schema

- Lossless transformations are the formal foundation of Database Design, of Database Normalisation, and of Database Reverse Engineering.
- Theorem: given an arbitrary database schema, there is a unique equivalent Canonical Abstract Relational Schema (CARM) equivalent, and it corresponds to its 6^{th} normal form with the careful addition of object identifiers.
- The CARM of a database schema corresponds to its cognitively plausible conceptual schema.

Queries and Updates

- Q(y) :- $\{$y|Employee(x) $\wedge$ name(x,y) $\wedge$
  Works-for(x,z) $\wedge$ name(z,"Project1")$\}$
- DELETE: $\exists$x.Project(x) $\wedge$ ProjCode(x,"P1")



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

END OF PART ONE

Formal definition of a DBox

- DBox $\mathcal{D}$: a set of database tuples of the forms $P : \langle a_1 \cdots a_n \rangle$
- DBox predicates: the database predicates P appearing in $\mathcal{D}$
- active domain $act(\mathcal{D})$: the constants a_i appearing in $\mathcal{D}$

Formal definition of a DBox

- DBox $\mathcal{D}$: a set of database tuples of the forms $P : \langle a_1 \cdots a_n \rangle$
- DBox predicates: the database predicates P appearing in $\mathcal{D}$
- active domain $act(\mathcal{D})$: the constants a_i appearing in $\mathcal{D}$
- An interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ embeds a database $\mathcal{D}$ iff
  - $a^{\mathcal{I}} = a$ for every $a \in act(\mathcal{D})$,
  - for every DBox predicate P and every $(u_1 \cdots u_n) \in \Delta^{\mathcal{I}} \times \cdots \times \Delta^{\mathcal{I}}$, $\langle u_1 \cdots u_n \rangle \in P^{\mathcal{I}}$ iff $P : \langle u_1 \cdots u_n \rangle \in \mathcal{D}$
- In other words, in every interpretation embedding $\mathcal{D}$ the extensions of the DBox predicates are given by the contents of the DBox, and are always the same

Formal definition of a DBox

- DBox $\mathcal{D}$: a set of database tuples of the forms $P : \langle a_1 \cdots a_n \rangle$
- DBox predicates: the database predicates P appearing in $\mathcal{D}$
- active domain $act(\mathcal{D})$: the constants a_i appearing in $\mathcal{D}$
- An interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ embeds a database $\mathcal{D}$ iff
  - $a^{\mathcal{I}} = a$ for every $a \in act(\mathcal{D})$,
  - for every DBox predicate P and every $(u_1 \cdots u_n) \in \Delta^{\mathcal{I}} \times \cdots \times \Delta^{\mathcal{I}}$, $\langle u_1 \cdots u_n \rangle \in P^{\mathcal{I}}$ iff $P : \langle u_1 \cdots u_n \rangle \in \mathcal{D}$
- In other words, in every interpretation embedding $\mathcal{D}$ the extensions of the DBox predicates are given by the contents of the DBox, and are always the same
- A model of a conceptual schema (FOL/DL) with a DBox $(\mathcal{T}, \mathcal{D})$ should always embed the DBox.

Determinacy - or implicit definability

Given a conceptual schema and a DBox, a query which depends only on the DBox predicates is called implicitly definable or determined by the DBox.

Definition (Implicit definability)

A query Q is *implicitly definable* from the DBox predicates in a theory $\mathcal{T}$ iff any two models of $\mathcal{T}$ that have the same domain and agree in what they assign to the DBox predicates also agree in what they assign to the query Q.

If a query is implicitly definable, then its evaluation depends only on the database; therefore implicitly definable queries characterise precisely views.

Beth (1953) and Craig (1957) constructively proved that there is a way to check whether an arbitrary query is determined by a DBox given a *general first-order logic* conceptual schema.

Example: implicit definability

$\texttt{Female} \subseteq \texttt{Person}$

$\texttt{Male} \subseteq \texttt{Person}$

$\texttt{Female} \cap \texttt{Male} = \emptyset$

$\texttt{Person} \subseteq \texttt{Male} \cup \texttt{Female}$

`Male` is implicitly definable from `Person` and `Female`

Rewriting - or explicit definability

If a query is implicitly definable given a general first-order logic conceptual schema, Beth (1953) and Craig (1957) constructively proved that it is possible to rewrite the query using only the vocabulary of the DBox (and therefore independent on the conceptual schema, since the extension of the DBox is fixed).

This is its explicit definition.

Example: explicit definability

$\texttt{Female} \subseteq \texttt{Person}$
$\texttt{Male} \subseteq \texttt{Person}$
$\texttt{Female} \cap \texttt{Male} = \emptyset$
$\texttt{Person} \subseteq \texttt{Male} \cup \texttt{Female}$

Male is implicitly definable from Person and Female;
the explicit definition is:

$\texttt{Male} = \texttt{Person} \setminus \texttt{Female}$

Problem: "unsafe" rewritings

Conceptual schema: $\texttt{Male} \doteq \neg\texttt{Female}$

DBox: $\texttt{Female} = \{\texttt{mary}\}$

Query: $\texttt{Q} \texttt{ :- } \neg\exists x.\texttt{Male}(x)$

Problem: "unsafe" rewritings

Conceptual schema: $\texttt{Male} \doteq \neg\texttt{Female}$
DBox: $\texttt{Female} = \{\texttt{mary}\}$
Query: $\texttt{Q} \texttt{ :- } \neg\exists x.\texttt{Male}(x)$

- If the only known individual is `mary`, the answer is YES.
- Indeed, this is the rewriting of the query: $\forall x.\texttt{Female}(x)$

Problem: "unsafe" rewritings

Conceptual schema: $\texttt{Male} \doteq \neg\texttt{Female}$
DBox: $\texttt{Female} = \{\texttt{mary}\}$ $\texttt{Car} = \{\texttt{herbie}\}$
Query: $\texttt{Q} \texttt{ :- } \neg\exists x.\texttt{Male}(x)$

- If the only known individual is `mary`, the answer is YES.
- Indeed, this is the rewriting of the query: $\forall x.\texttt{Female}(x)$
- But if we add to the domain the individual `herbie` as a `car` (namely, we extend the active domain), then the answer is NO.

Problem: "unsafe" rewritings

Conceptual schema: $\texttt{Male} \doteq \neg\texttt{Female}$
DBox: $\texttt{Female} = \{\texttt{mary}\}$ $\texttt{Car} = \{\texttt{herbie}\}$
Query: $\texttt{Q} \texttt{ :- } \neg\exists x.\texttt{Male}(x)$

- If the only known individual is `mary`, the answer is YES.
- Indeed, this is the rewriting of the query: $\forall x.\texttt{Female}(x)$
- But if we add to the domain the individual `herbie` as a `car` (namely, we extend the active domain), then the answer is NO.

The query is not domain independent.

Domain independent formulas

ϕ is domain independent whenever:

given $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ and $\mathcal{J} = \langle \Delta^{\mathcal{J}}, \cdot^{\mathcal{I}} \rangle$

with $\Delta^{\mathcal{I}} \subseteq \Delta^{\mathcal{J}}$

then $\langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle \models \phi$ iff $\langle \Delta^{\mathcal{J}}, \cdot^{\mathcal{I}} \rangle \models \phi$

Observations:

- E/R, ORM, UML are all domain independent languages
- Relational algebra is exactly the domain independent fragment of first order logic
- (Core) SQL is equivalent to relational algebra

Domain independent rewritings

Theorem

If a first-order logic conceptual schema is domain independent, then the rewriting of determined domain independent first-order logic queries is also a domain independent first-order logic query.

Theorem

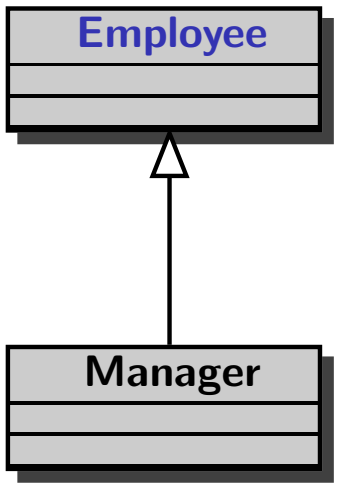
The domain independent fragments of the decidable description logics $\mathcal{ALCHOI}$ and $\mathcal{ALCHOQ}$ are equally expressive to their syntactic safe-range fragments, and they have finitely controllable determinacy.

Task: query rewriting

Our goals are

1. to check whether the answers to a given domain independent query under a domain independent first-order logic conceptual schema are *solely* determined by the DBox and, if so,
2. to find a first-order/SQL *equivalent* (modulo the conceptual schema) domain independent rewriting (called exact rewriting) of the query in terms of the DBox predicates to allow the use of standard database technology for answering the query.
3. Indeed, this means we benefit from the low computational complexity – logarithmic space in the size of the data – of answering first-order queries on relational databases.
4. If queries are just atomic queries, then it is possible to pre-compute all the rewritings of all the determined predicates as relational views, and to allow arbitrary SQL queries on top of them.

Conceptual Schema Abduction



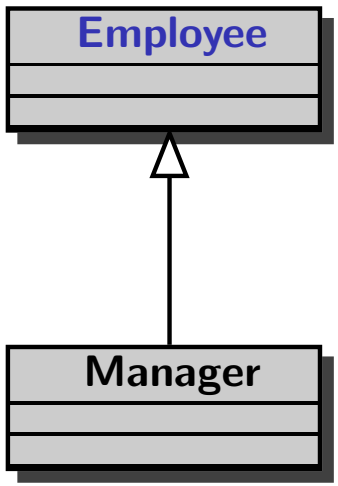

```
Manager = { John, Paul }
```

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

Exact rewriting of Q:

Certain answer of Q:

Conceptual Schema Abduction



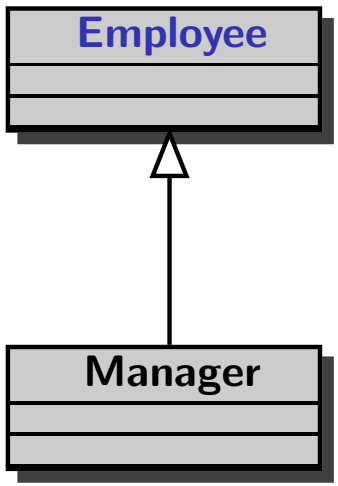

```
Manager = { John, Paul }
```

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

Exact rewriting of Q: impossible.

Certain answer of Q:

Conceptual Schema Abduction

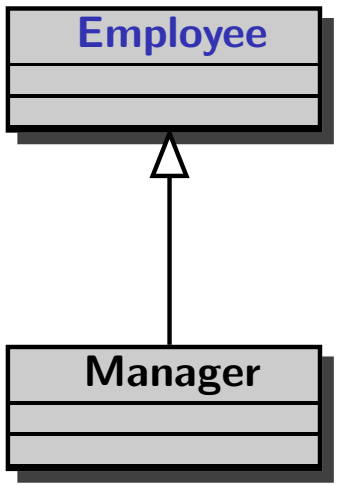


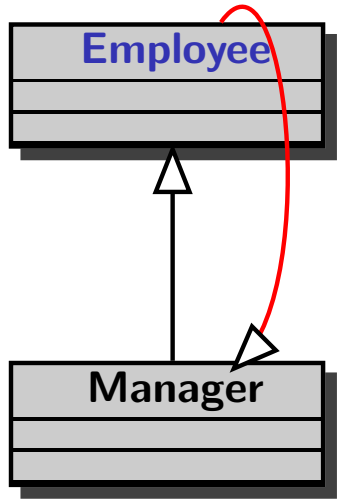

```
Manager = { John, Paul }

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

Exact rewriting of Q: impossible.

Certain answer of Q: {John, Paul}

Conceptual Schema Abduction





`Manager = { John, Paul }`

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

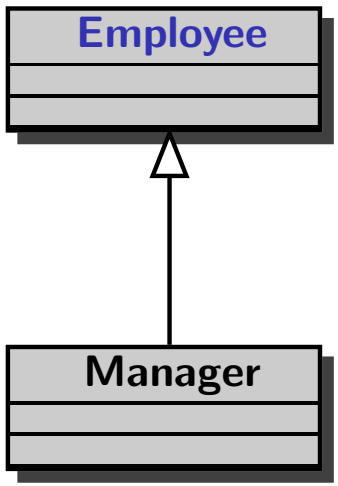
Exact rewriting of `Q`: impossible.
Certain answer of `Q`: {John, Paul}

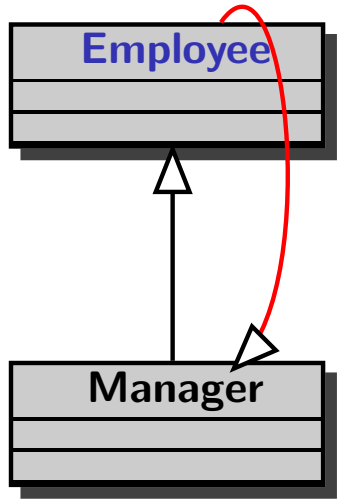
`Manager = {John, Paul}`

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

Exact rewriting: `Manager(X)`
Certain answer of `Q`: {John, Paul}

Conceptual Schema Abduction





`Manager = { John, Paul }`

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

Exact rewriting of `Q`: impossible.
Certain answer of `Q`: {John, Paul}

`Manager = {John, Paul}`

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

Exact rewriting: `Manager(X)`
Certain answer of `Q`: {John, Paul}

The abduction is the model-theoretically *least committing* extension of the ontology such that the query becomes implicitly definable.

NULL values in SQL: algebra and logic

Spouse = {"John", "Mary"}
Spouse = {"Louise", NULL}

```
SELECT 1,2 FROM Spouse WHERE 1 = 1 AND 2 = 2;
```

$\sigma_{1=1,2=2}$(Spouse $\times$ Spouse)

{x,y|Spouse(x,y) $\wedge$ Spouse(x,y)}

SQL answer: {"John", "Mary"}

NULL values in SQL: algebra and logic

Theorem (extended Codd's theorem with SQL null values)

The following are equivalent:

- *standard core SQL with null values over a three-valued logic;*
- *SQL with null values over classical two-valued logic;*
- *a simple extension with null values of standard relational algebra in which the selection operators fails systematically the comparisons involving null values;*
- *a first-order logic language with an explicit NULL term, with the interpretation of predicates as partial tuples (i.e., predicates denote tuples over subsets of the arguments instead of the whole set of arguments); in this logic the null value does not appear as an element of the semantic domain.*
- *a classical first order logic with an exponentially larger signature.*

Conclusions

Conclusions

Do you want to use knowledge representation
with databases?

Conclusions

Do you want to use knowledge representation
with databases?

Pay attention!